Tema 3

Sistemas Digitales

EdC-ISW

Contenidos del Tema 2

1. El nivel RT (*Register Transfer*)
2. Diseño de Sistemas Digitales
3. Diseño de la unidad de datos
   - Interconexión mediante buses
   - Ejemplo: diseño de una calculadora simple
4. Diseño de la unidad de control:
   - Descripción mediante cartas ASM
   - Descripción mediante HDL Verilog **(LAB)**

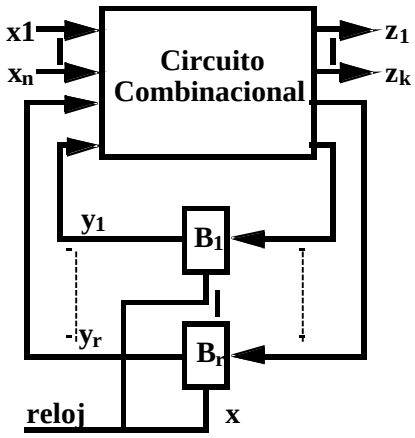
Contenidos del Tema 2 (cont.)

5. Ejemplo 1: Calculadora simple con dos registros
6. Ejemplo 2: Calculara simple (solución 3 buses)
7. Ejemplo 3: Calculadora simple con 8 registros
8. De la calculadora al computador

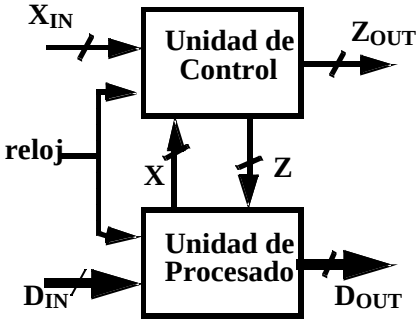
Contenidos del Tema 2

1. ***El nivel RT (Register Transfer)***
   - Descripción de componentes
   - Operaciones entre registros
2. Diseño de Sistemas Digitales
3. Diseño de la Unidad de Datos
4. Diseño de la Unidad de Control
5. Calculadora simple con dos registros
6. Calculadora simple (solución 3 buses) (cont.)

Nivel RT: circuitos vs sistemas



CIRCUITOS		SISTEMAS DIGITALES
0,1	Información	Palabras de datos
De conmutación	Nivel/Lenguaje	RT (Register Transfer)
Máquina de estados	Funcionalidad	Instrucciones/ cartas ASM
Puertas y biestables	Componentes	MUX, ALU, ..., registros,...
Líneas (cables)	Conexión	Buses
Combinacional y almacenamiento (memoria)	Organización	Procesado de datos y control

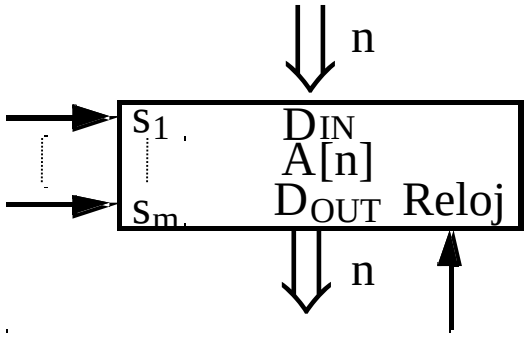


X : cualificadores o entradas de control
Z : comandos o salidas de control
D: datos

Nivel RT: Descripción de componentes

- Registro: unidad básica de almacenamiento de datos

Representación estructural:



Control S_1 S_2 ....S_m	Operación a Nivel RT
0 0 ... 0	$A \leftarrow D_{IN}$
0 0 ... 1	$A \leftarrow 0$
...	...
1 1 ... 1	otras

(Descripción funcional)

Ejemplos de operaciones en Registros

- De escritura (secuenciales)

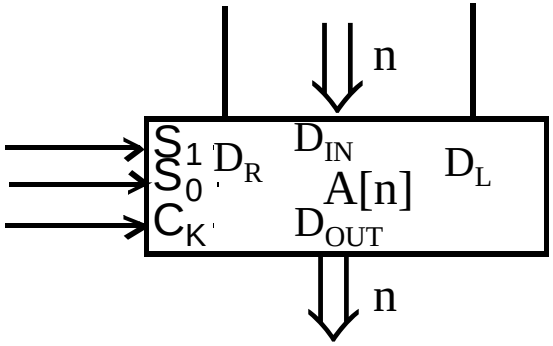
Operación	Notación RT
Carga en paralelo	$A \leftarrow D_{IN}$
Despl. izquierda	$A \leftarrow SHL(A,D_L)$
Despl. derecha	$A \leftarrow SHR(A,D_R)$
Incremento	$A \leftarrow A + 1$
Decremento	$A \leftarrow A - 1$
Puesta a 0	$A \leftarrow 0$
Puesta a 1	$A \leftarrow 1\ldots1$
NOP/Inhibición	$A \leftarrow [A]$

- De lectura (combinacionales)

Operación	Notación RT
Lectura incondicional	Dout = [A]
Lectura condicional	R: Dout = [A]
Función del dato	Z=1 si [A]=0

Ejemplo 1:
Descripción RT del Registro Universal de n bits

- Representación estructural

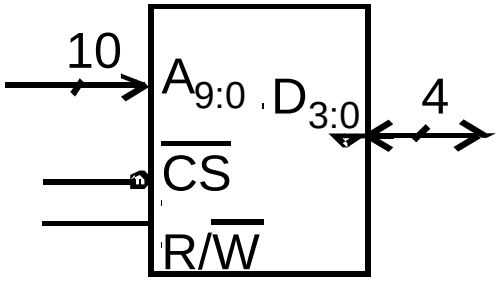


- Descripción funcional

Control S_1 S_0	Escritura A ←	Lectura D_{OUT}
0 0	A ← A	D_{OUT}=[A]
0 1	A ← D_{IN}	
1 0	A ← SHR(A,D_R)	
1 1	A ← SHL(A,D_L)	

Ejemplo 2:
Descripción RT de la RAM comercial 2114

- Representación estructural



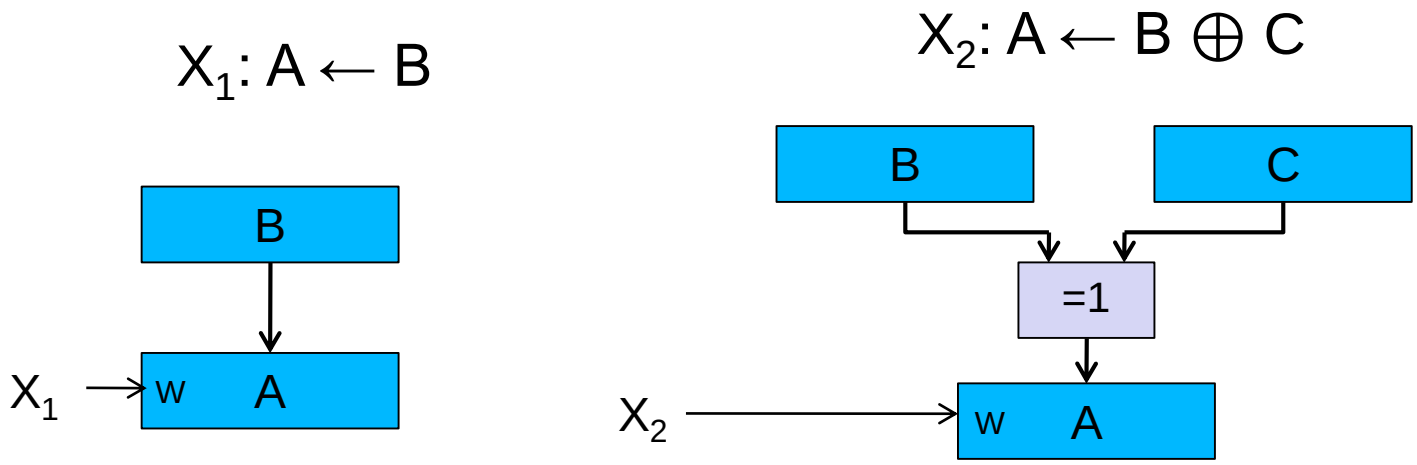
RAM 1Kx4

- Representación funcional

Control CS' R/W'		Escritura RAM ←	Lectura D_{3-0}=
1	-	RAM ← RAM	HI
0	1	RAM← RAM	[RAM(A)]
0	0	RAM(A) ←D3-0	D3-0

Nivel RT: Operaciones entre varios registros

- Ejemplos:



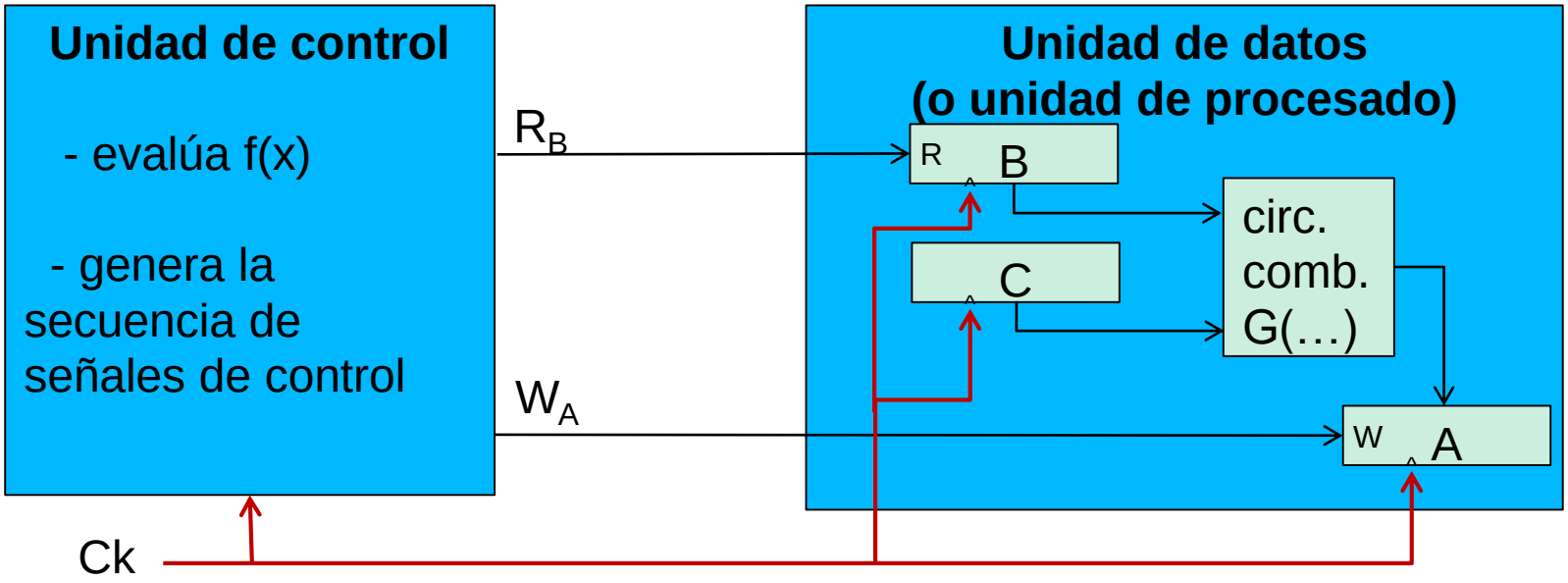
Contenidos del Tema 2

1. El nivel RT (*Register Transfer*)
2. ***Diseño de Sistemas Digitales***
   - Estructura general de un Sistema Digital
   - Macrooperaciones y microoperaciones
   - Metodología de Diseño de Sistemas Digitales
3. Diseño de la Unidad de Datos
4. Diseño de la Unidad de Control

(cont.)

Estructura general del sistema digital

- Generalización: f(x): A ← G(B, C, …)

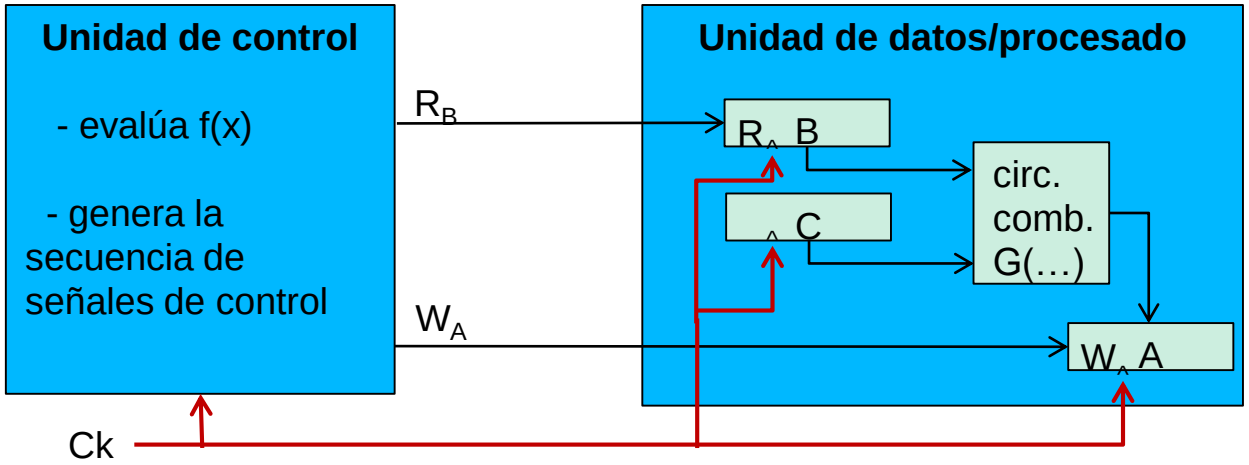


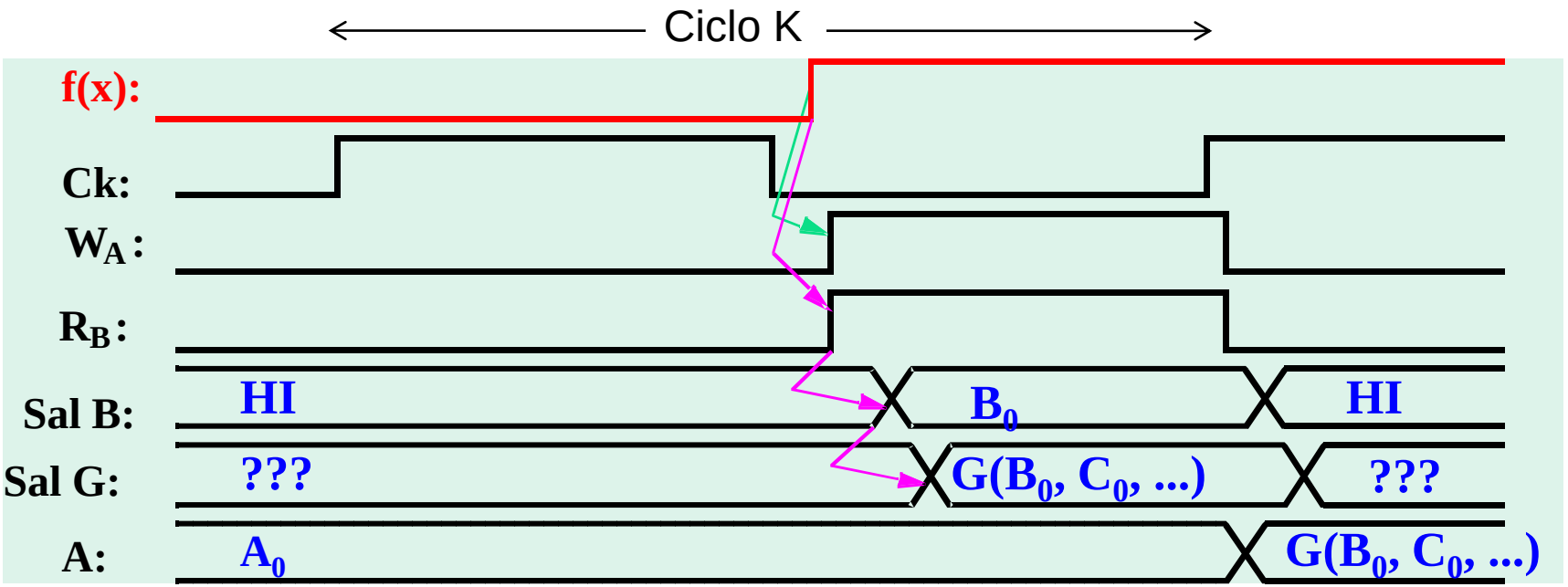
Nivel RT: Macro y micro - operaciones

- **Macrooperación (o instrucción):**
  - Es cada tarea que especifica el usuario y que el sistema realiza automáticamente
  - En general, el sistema emplea **varios ciclos** en su ejecución.
  - La unidad de control "dirige/supervisa" la tarea realizada
- **Microoperación (μop):**
  - Es cada tarea que el sistema realiza en un **único ciclo** de reloj
  - En general, consiste en una o varias transferencias entre registros

Nivel RT: Ejecución de una μop

Ciclo K
f(x): A ← G(B, C, …)





Metodología de diseño de Sist. Digitales

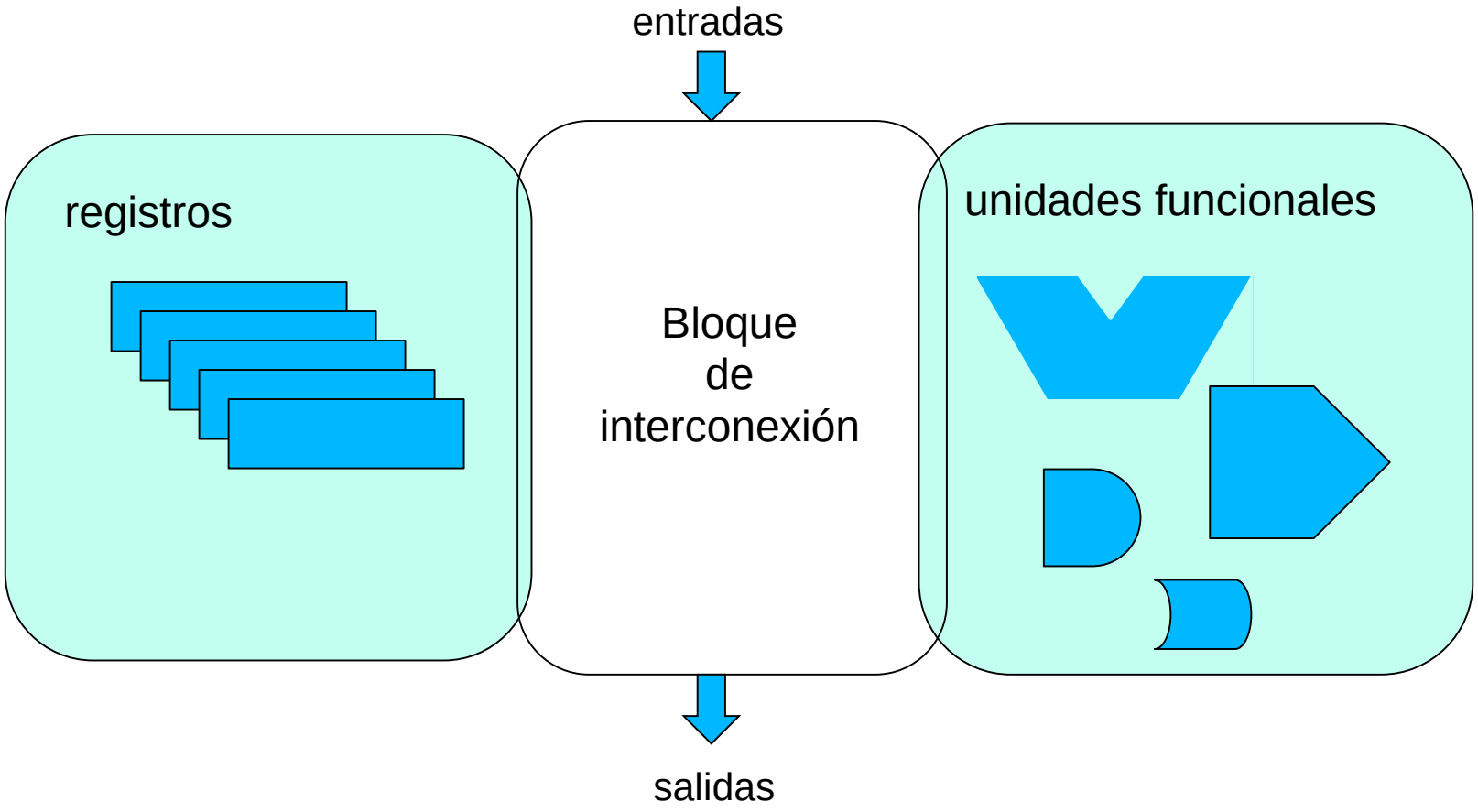
- Paso 1: Comprender claramente las especificaciones del sistema a diseñar y definir el conjunto de instrucciones y/o operaciones.
- Paso 2: Proponer una Unidad de datos capaz de ejecutar todas las operaciones especificadas. (Debe incluir los registros "visibles" referenciados en la macrooperaciones).
- Paso 3: Describir todos los componentes a nivel RT estructural y funcional.
- Paso 4: Descomponer las macrooperaciones en microoperaciones para la arquitectura propuesta.
- Paso 5: Descripción de la U. Control mediante carta ASM
- Paso 7: Descripción HDL de la U. Datos y U. de Control **(LAB)**

Contenidos del Tema 2

1. El nivel RT (Register Transfer)
2. Diseño de Sistemas Digitales
3. ***Diseño de la Unidad de Datos***
   - Componentes
   - Bloque de interconexión
   - Arquitecturas genéricas basadas en ALU
4. Diseño de la Unidad de Control

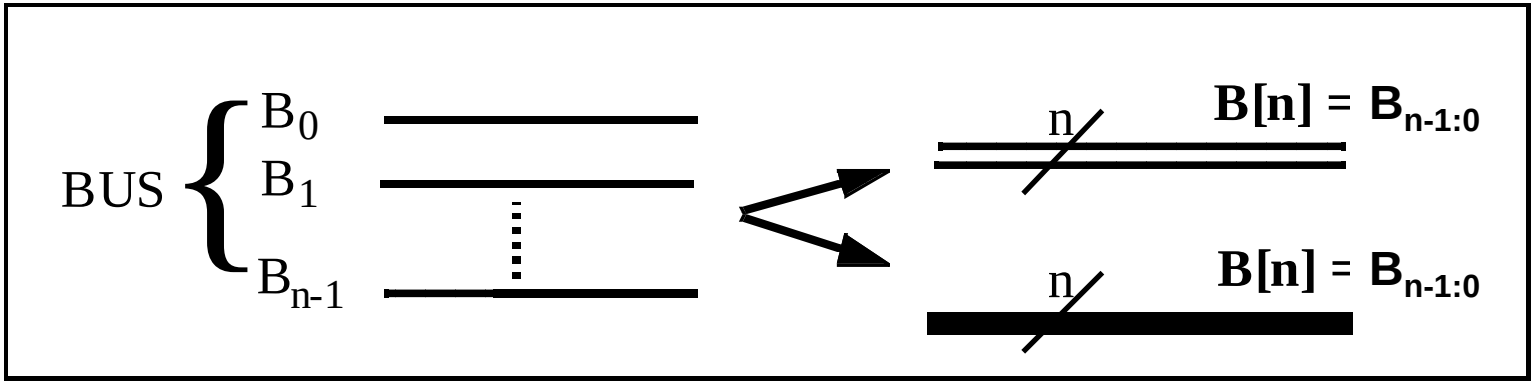
(cont.)

Componentes de la Unidad de Datos

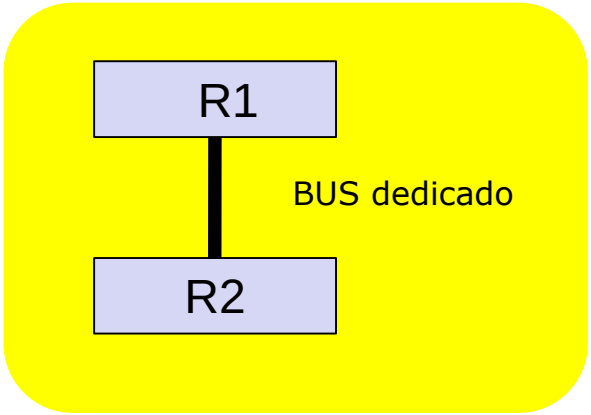


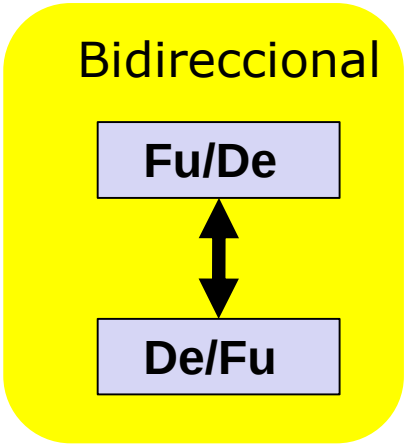
Bloque de interconexión: buses

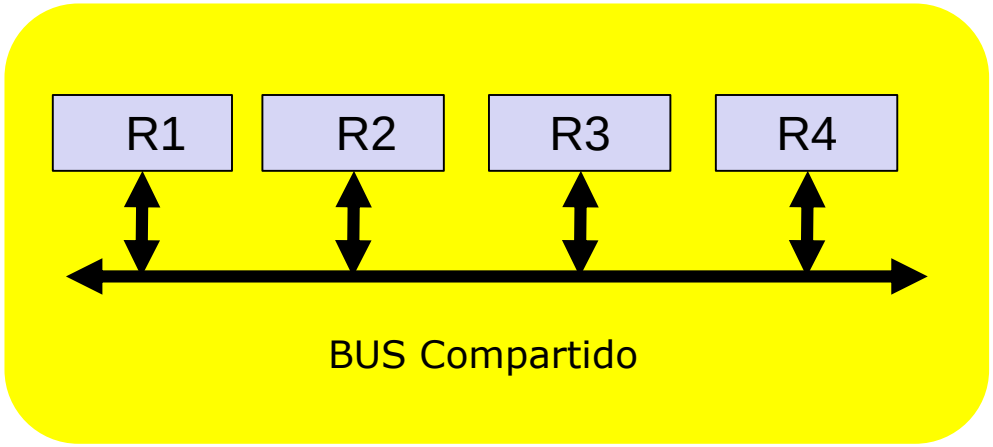
- Bus:
  en un sistema digital, un bus es un conjunto de *n* líneas ordenadas que discurren en paralelo y transportan información (palabras)



Bloque de interconexión: buses







¿Cómo interconectar registros?

Ejemplo:

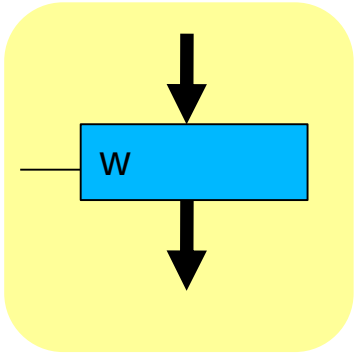
Se dispone de 4 registros A_3,A_2,A_1,A_0 con carga en paralelo.

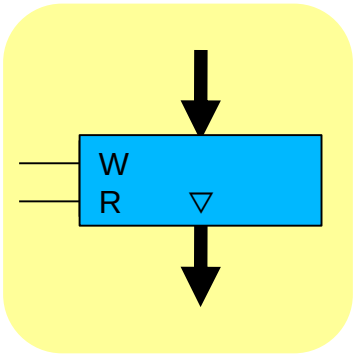
- Hay que realizar la conexión para la transferencia $A_F \rightarrow A_D$, con F, D $\in$ {0, 1, 2, 3}

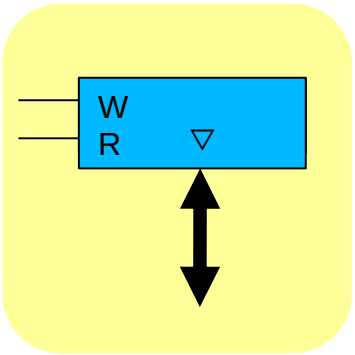
- Selección de fuente: F_1F_0
- Selección de destino: D_1D_0

Transferencia entre registros (tres soluciones)

- Caso 1: registros con salida y entrada separadas
- Caso 2: registros con salida y entrada separadas, salida triestado
- Caso 3: registros con un único bus bidireccional de salida y entrada

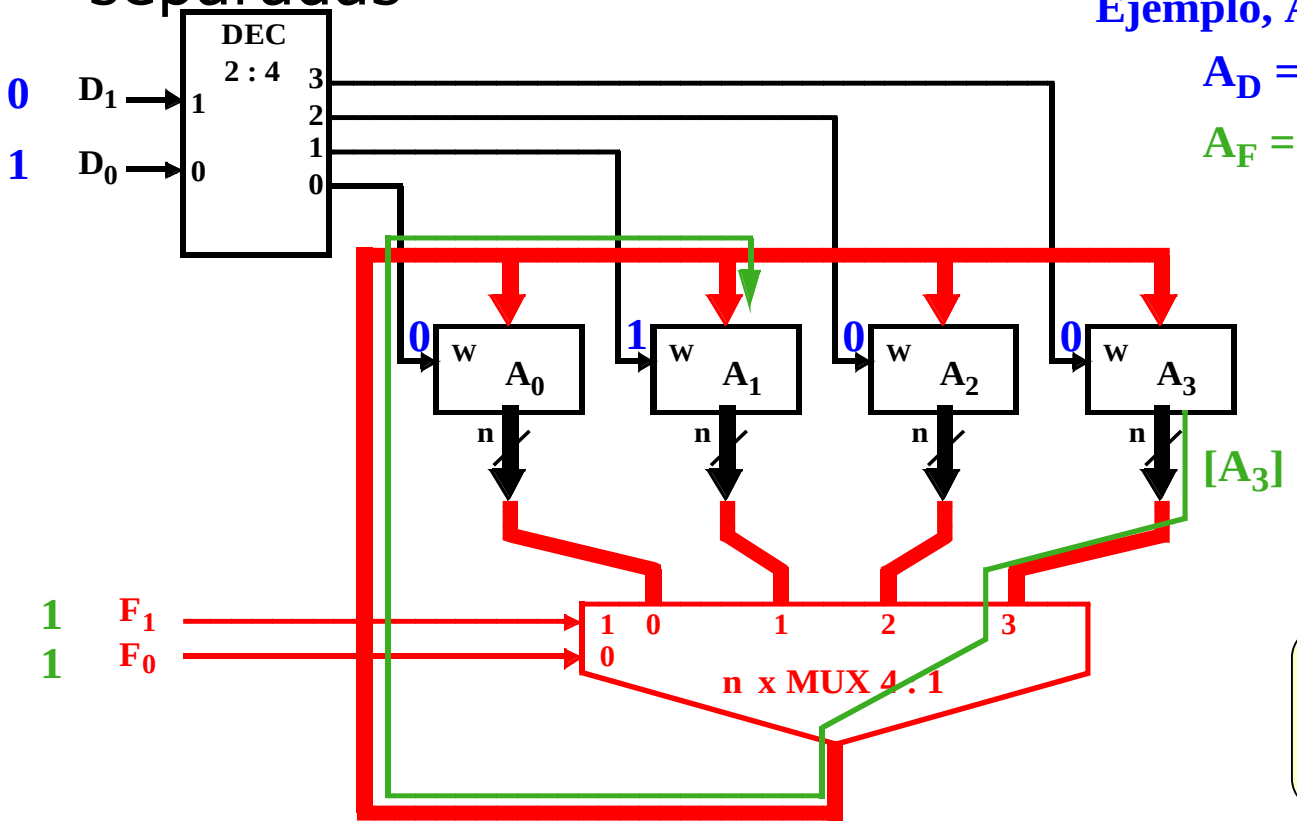






Transferencia entre registros (solución 1)

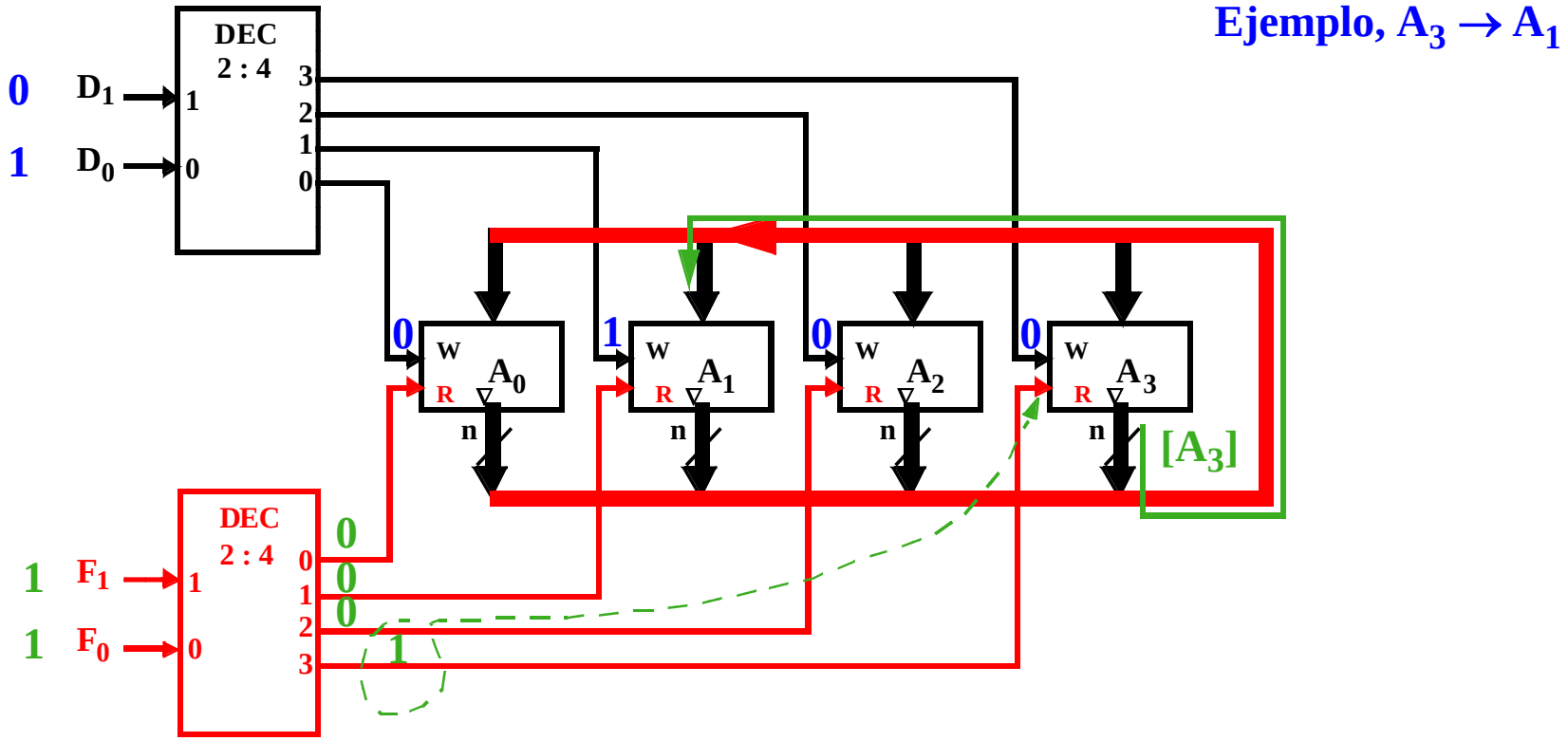
- Caso 1: registros con salida y entrada separadas



Solución multiplexada

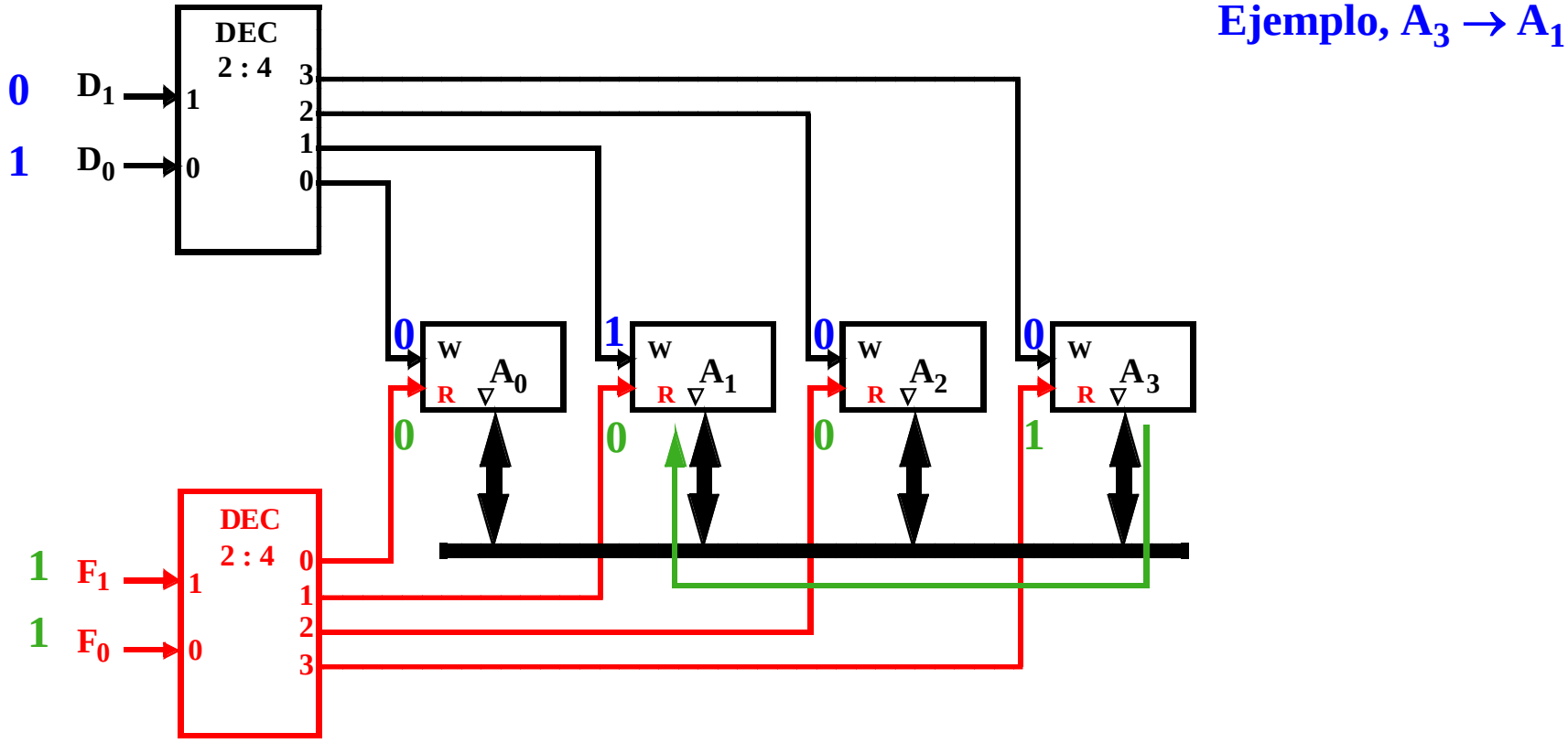
Transferencia entre registros (solución 2)

- Caso 2: Registros con entrada y salida separadas (salidas tri-estado)



Transferencia entre registros (solución 3)

- Caso 3: Entrada/salidas bidireccionales (conexión a través de un Bus compartido)



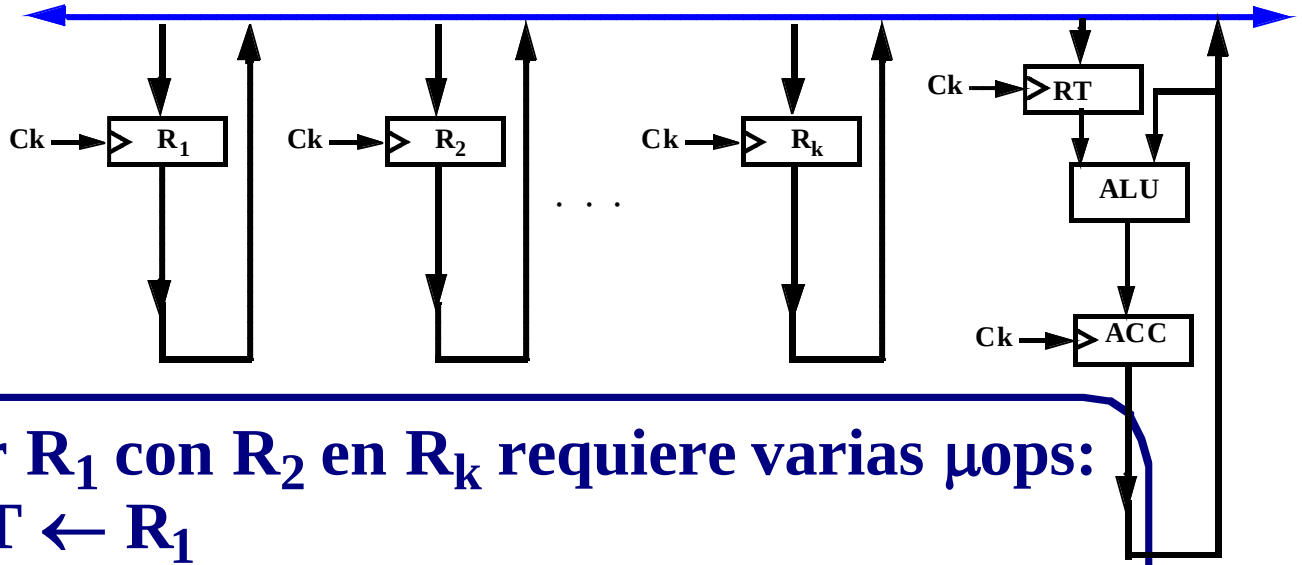
Arquitecturas genéricas basadas en ALU

- Ejemplo: Proponga una arquitectura de una unidad de datos que permita la realización de la macrooperación $R_k \leftarrow R_1 + R_2$

  - Caso 1: bus simple
  - Caso 2: doble bus
  - Caso 3: triple bus

Arquitecturas genéricas basadas en ALU

- Caso 1: bus simple



Sumar R_1 con R_2 en R_k requiere varias μops:

1 $RT \leftarrow R_1$

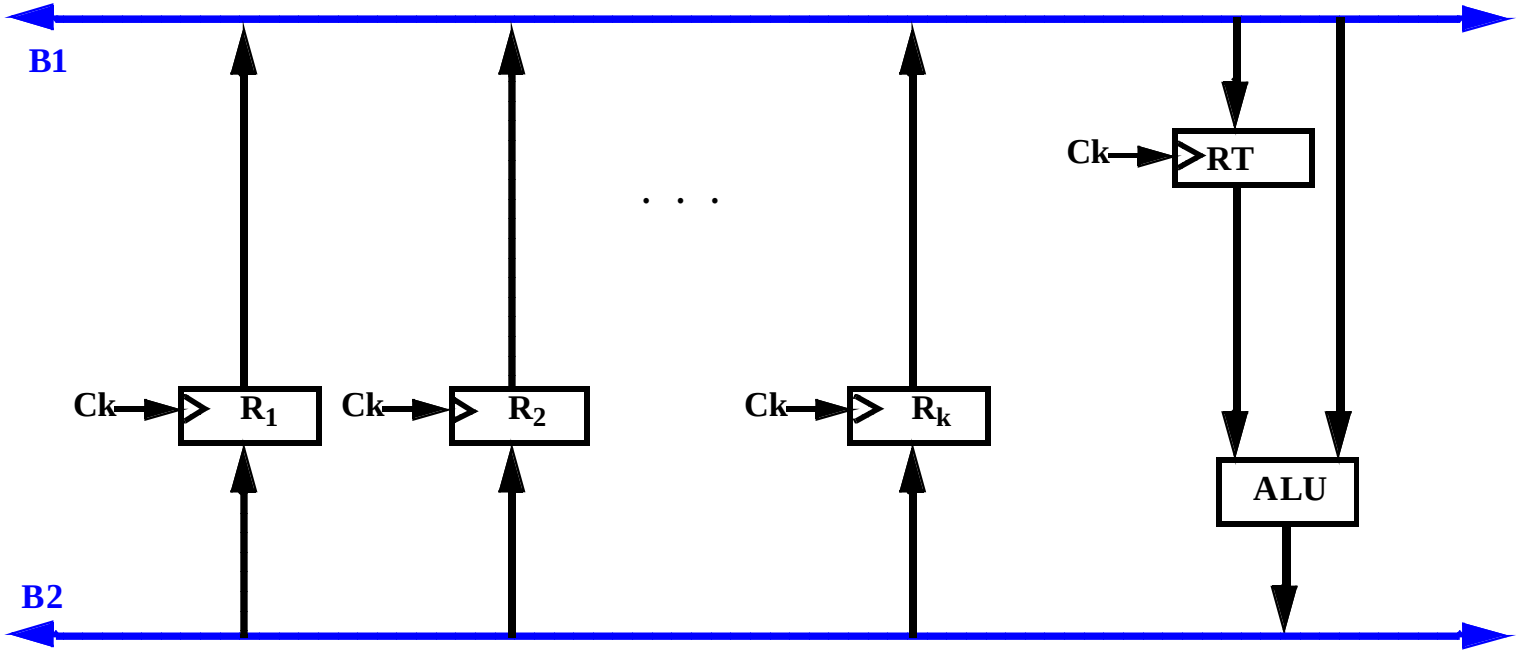
2 $ACC \leftarrow RT + R_2$

3 $R_k \leftarrow ACC$

- Ri debe tener salida condicional

Arquitecturas genéricas basadas en ALU

- Caso 2: doble bus



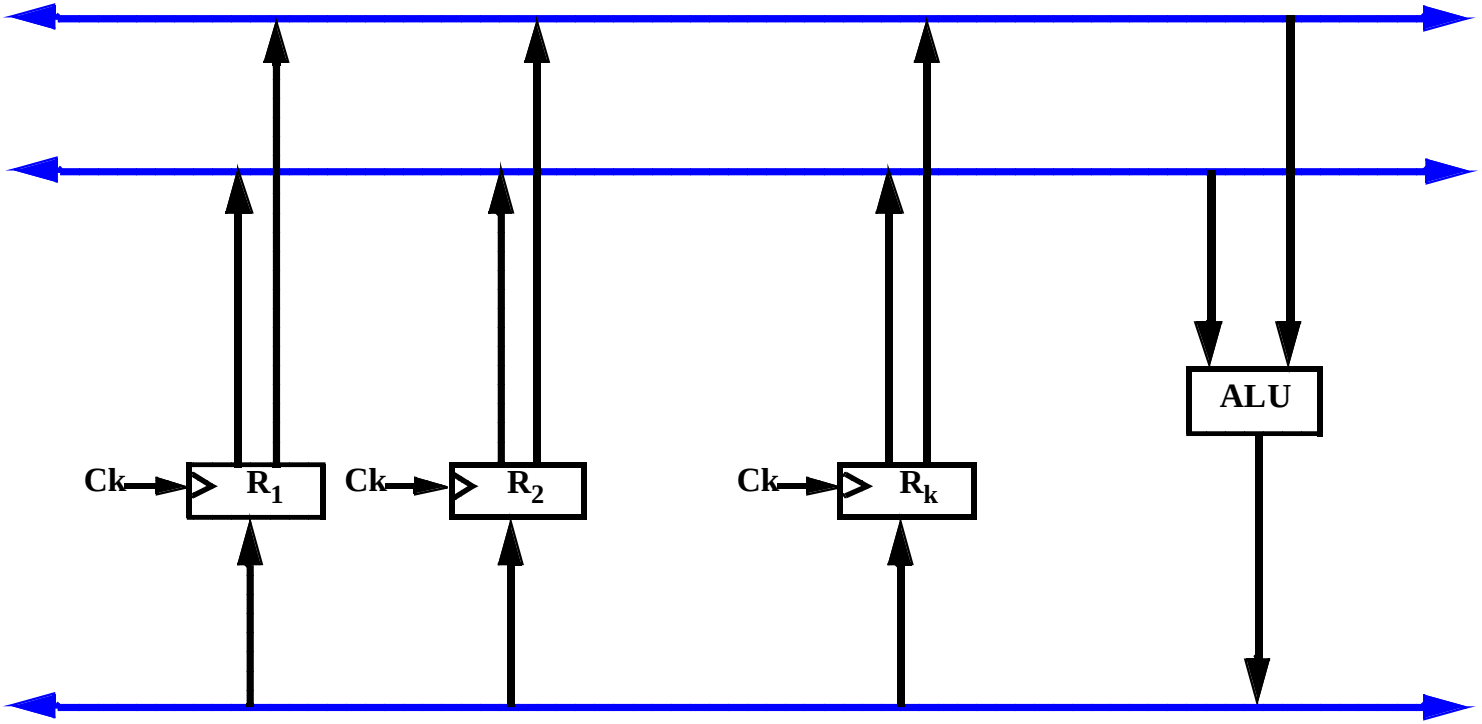
Sumar R_1 con R_2 en R_k requiere varias μops, pero menos:

1 $RT \leftarrow R_1$

2 $R_k \leftarrow RT + R_2$

Arquitecturas genéricas basadas en ALU

- Caso 3: triple bus



Sumar R_1 con R_2 en R_k requiere sólo 1 μop:

$R_k \leftarrow R_1 + R_2$

Contenidos del Tema 2

1. El nivel RT (*Register Transfer*)
2. Diseño de Sistemas Digitales
3. Diseño de la Unidad de Datos
4. ***Diseño de la Unidad de Control***
   - Descripción mediante cartas ASM
   - Técnicas de realización de U. de Control

(cont)

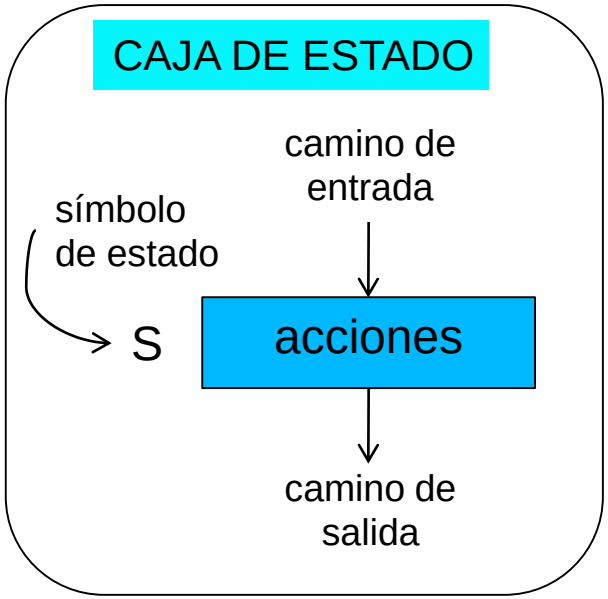
Descripción mediante cartas ASM

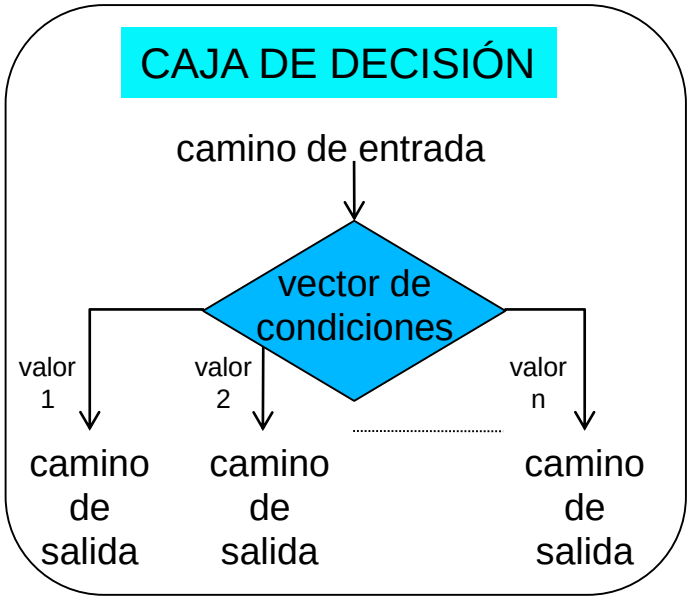
- Un método alternativo a los diagramas de estado para Sistemas Digitales con muchas entradas, muchas salidas y muchos estados
- ASM: ***Máquinas de Estado Algorítmicas***

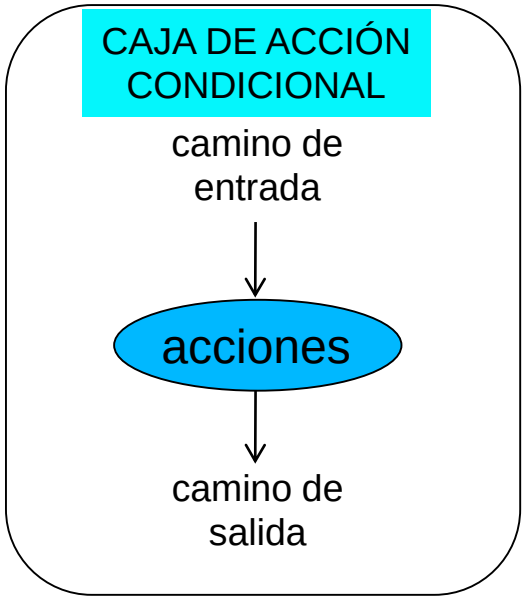
  (Algorithmic State Machines)

Descripción mediante cartas ASM

- Definiciones

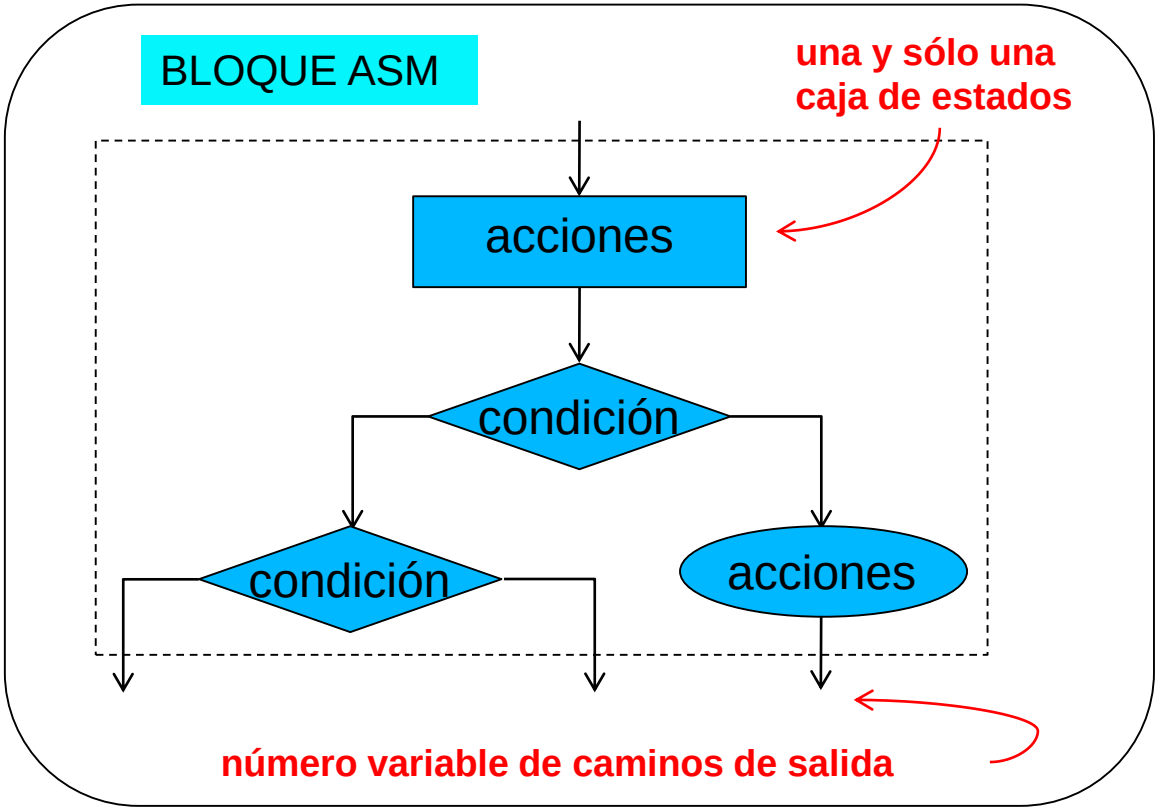






Descripción mediante cartas ASM

- Definiciones

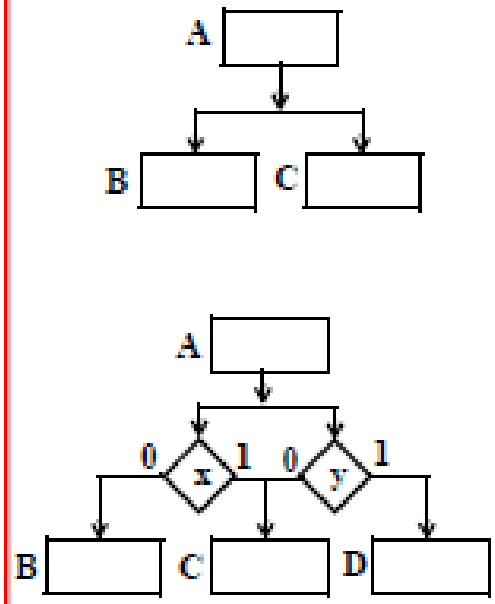


CARTA ASM

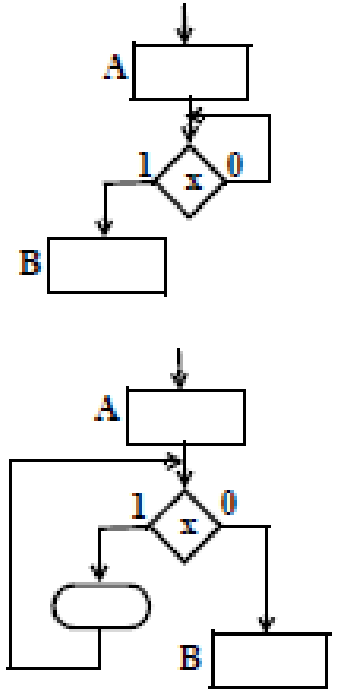
grafo orientado y cerrado que interconecta bloques ASM

Errores comunes en cartas ASM

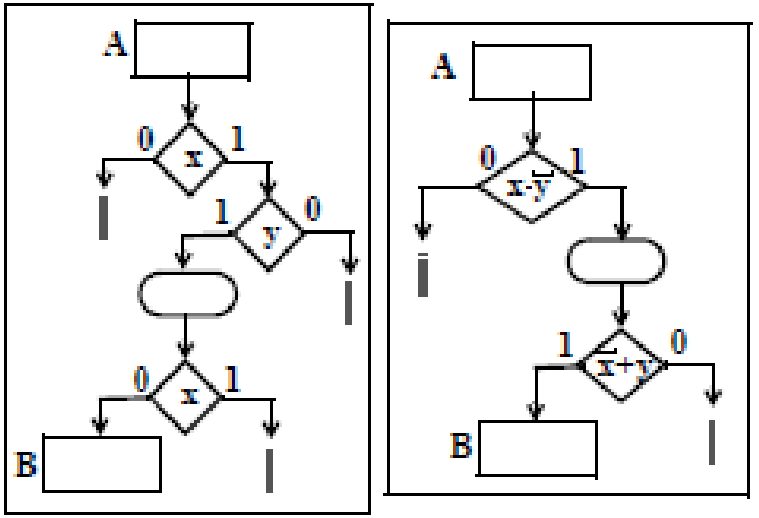
Próximo estado sin determinar



Cerrar lazos sin cajas de estado

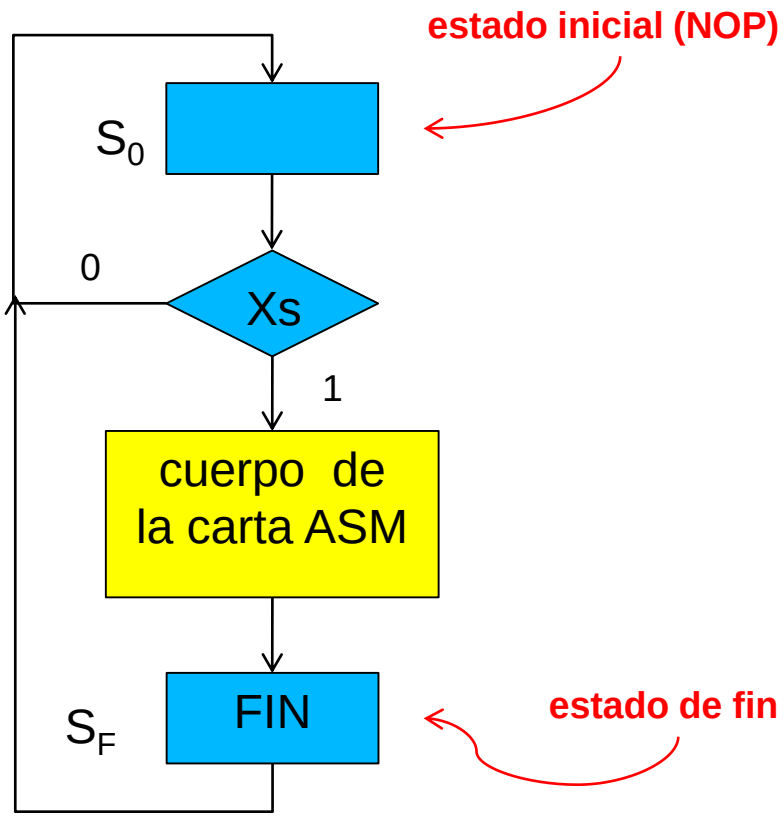


No garantizar la posibilidad lógica de todos los caminos



Descripción de un Sistema Digital mediante cartas ASM

- Inicio y fin de operación

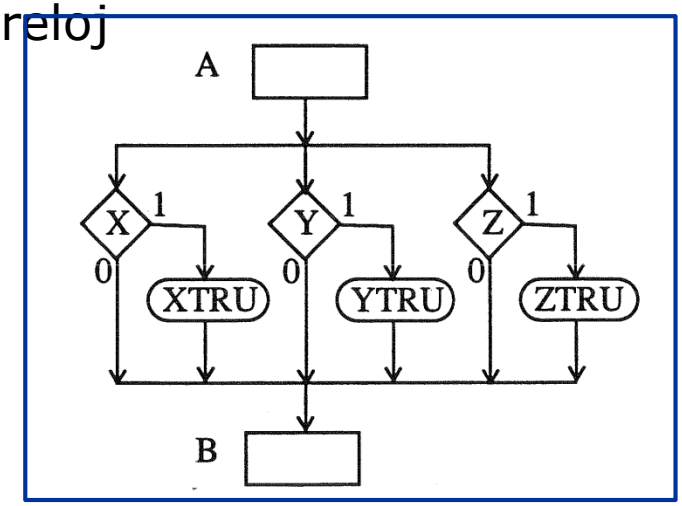


Xs: entrada con la que se inicia la operación (Xstart)

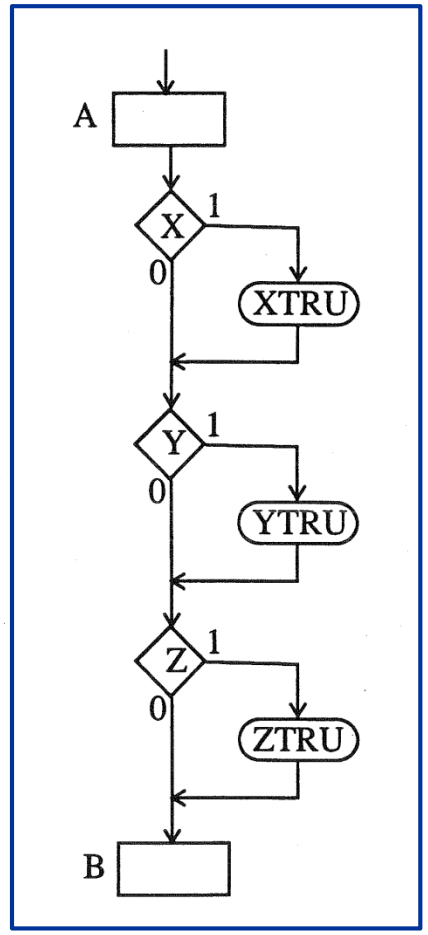
FIN: salida que indica que la operación ha terminado

Consideraciones temporales

- El orden de las cajas en un bloque ASM no implica orden temporal.
- Todas las tareas de un bloque ASM se hacen en un ciclo de reloj

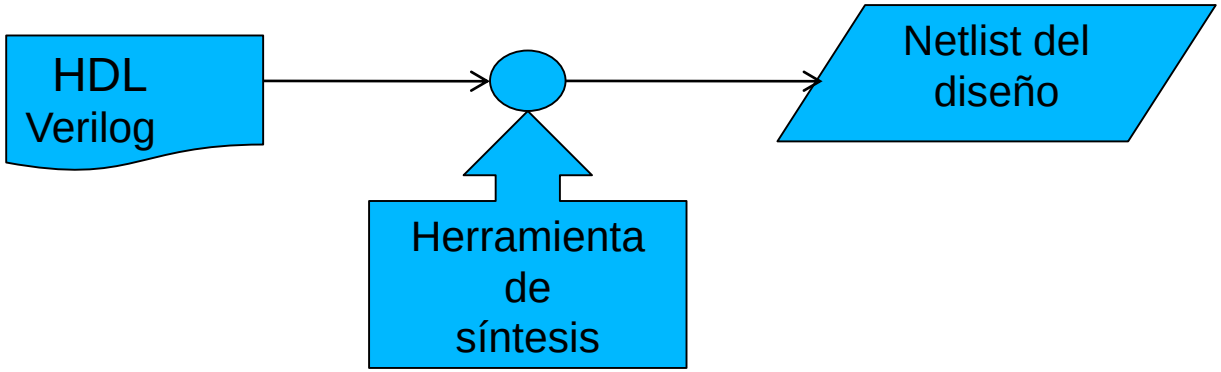


- Igual significado lógico



Técnicas de realización de Unidades de control

- Estrategias:
  - Un biestable por estado
  - Control Microprogramado (contador+ROM)
  - Mediante un PLD, a partir de la especificación HDL



EdC (Teoría y problemas): el objetivo es llegar sólo a la Carta ASM

EdC (Laboratorio): verilog e implementación en FPGA

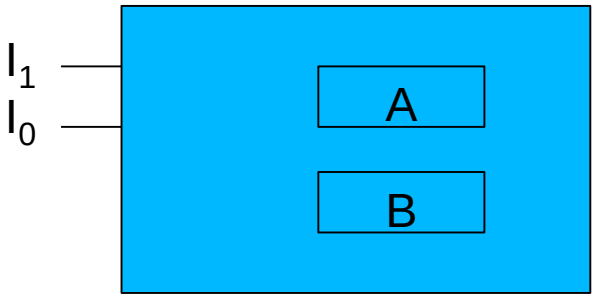
Contenidos del Tema 2

1. El nivel RT (*Register Transfer*)
2. Diseño de Sistemas Digitales
3. Diseño de la unidad de datos
4. Diseño de la unidad de control
5. ***Ejemplo 1: Diseño de una calculadora simple de 2 registros***
6. Ejemplo 2: Calculadora simple (solución 3 buses)
7. Ejemplo 3: Caluladora simple con 8 registros
8. De la Calculadora al computador

Diseño de una calculadora simple

- Paso 1- Especificaciones del sistema a diseñar:
  - ***Se dispone de 2 registros, A y B y se desea poder realizar cualquiera de las siguientes operaciones:***

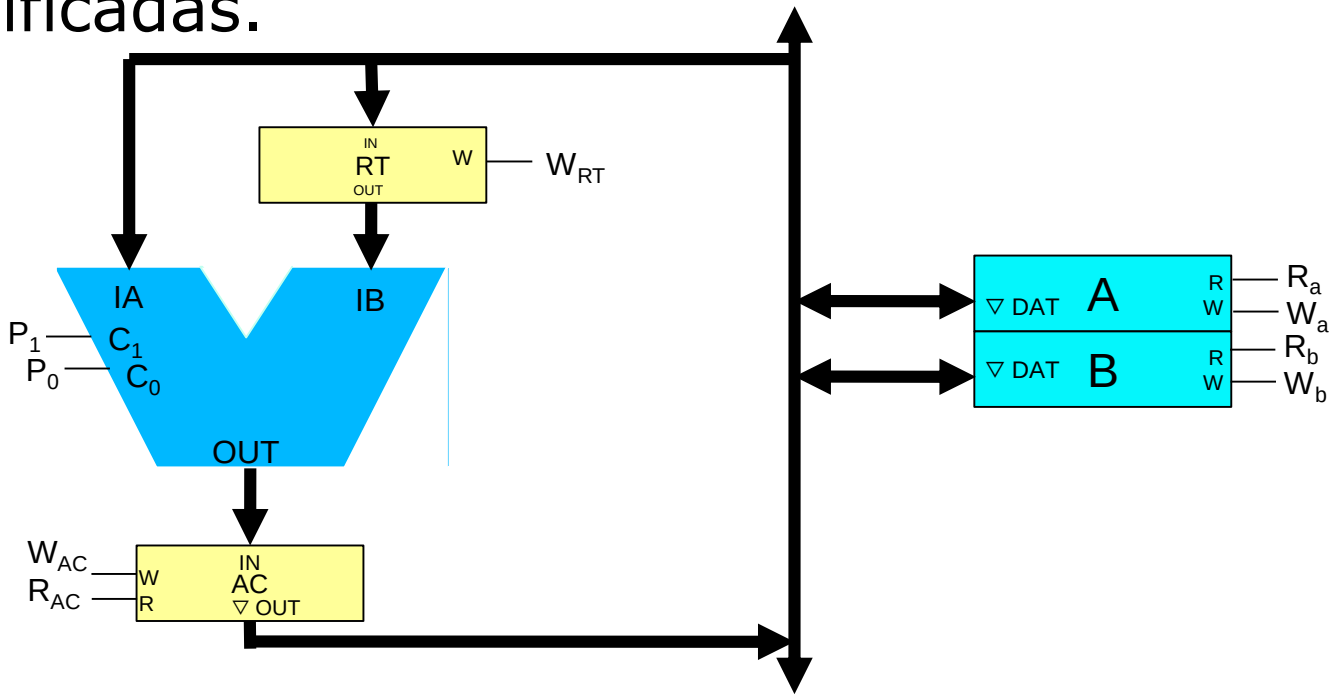
I_1 I_0	operación
0 0	A ← A + B
0 1	B ← A + B
1 0	A ← A - B
1 1	B ← A - B



  - Se han asignado los códigos de modo que el registro destino se identifica con I_0 y la operación con I_1.

Diseño de la unidad de datos de una calculadora

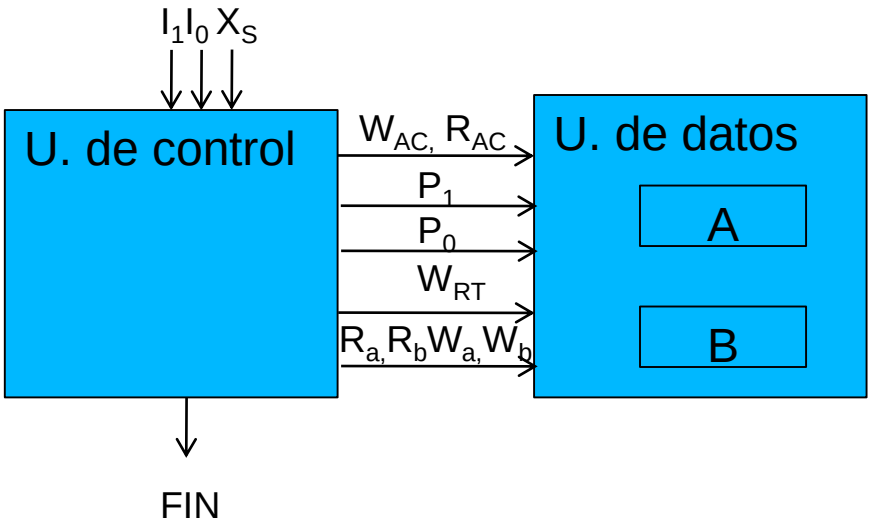
- Paso 2 - Proponemos una arquitectura genérica de un bus capaz de ejecutar las operaciones especificadas.



(la arquitectura no es única)

Diseño de la calculadora simple

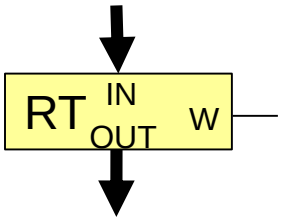
- Organización del sistema digital:



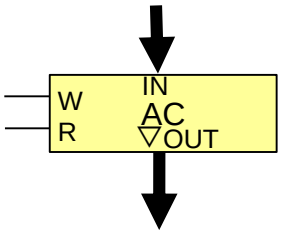
- El usuario especifica la operación proporcionando el valor de $I_{1,}$ I_0 y genera la orden de comienzo con X_S

Diseño de la unidad de datos de una calculadora

- Paso 3 – Describimos los componentes a nivel RT

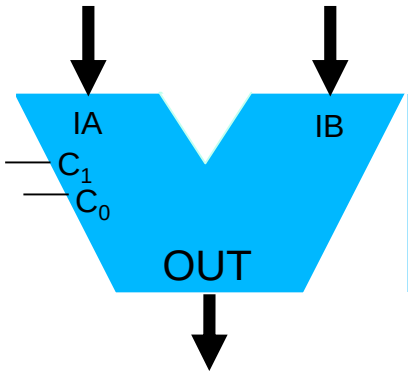


W	RT ←	OUT=
0	RT	[RT]
1	IN	[RT]



W R	AC ←	OUT=
0 0	AC	HI
0 1	AC	[AC]
1 0	IN	HI
1 1	IN	[AC]

RW	X ←	DAT=
0 0	X	HI
0 1	DAT	DAT
1 0	X	[X]
1 1	Proh	proh



C_1 C_0	OUT=
0 0	IA + IB
0 1	IA
1 0	IA –IB
1 1	IB

- Ojo: faltan los relojes de los elementos secuenciales

(Ejemplo de Descripción Verilog de la unidad de datos de la calculadora)

```
//declaración del tipo módulo correspondiente a RT
module type1 #(parameter width=8, initial_value=0)
                (input wire W, ck, input wire [width-1:0] IN, output reg [width-1:0] OUT=initial_value);
        always@(posedge ck)
                if(W)
                        OUT<=IN;
endmodule
//declaración del tipo módulo correspondiente a AC
module type2 #(parameter width=8, initial_value=0)
                (input wire W, R, ck, input wire [width-1:0] IN, output wire [width-1:0] OUT);
        wire [width-1:0] internal_bus;
        type1 #(width,initial_value) internal_reg(W,ck,IN,internal_bus);
        assign OUT = R ? internal_bus : 'bz;
endmodule
//declaración del tipo módulo correspondiente a RA y RB
module type3 #(parameter width=8, initial_value=0) (input wire W, R, ck, inout wire [width-1:0] DAT);
        type2 #(width,initial_value) internal_register(W,R,ck,DAT,DAT);
endmodule
//declaración del tipo módulo correspondiente a la ALU
module ALU_type #(parameter width=8) ( input wire C1, C0, input wire [width-1:0] IA,IB, output reg [width-1:0] OUT);
        always@(*)
                case({C1,C0})
                        2'b00: OUT=IA+IB;
                        2'b01: OUT=IA;
                        2'b10: OUT=IA-IB;
                        2'b11: OUT=IB;
                endcase
endmodule
```

```
//declaración de la unidad de procesado de datos
module unidad_datos #(parameter width=8, initial_A=0, initial_B=1)
            (input wire ck,WAC,RAC,WRT,Ra,Rb,Wa,Wb,P0,P1);
wire [width-1:0] common_bus, ALU_out, RT_out;
type1 #(.width(width)) RT(WRT,ck,common_bus,RT_out);
type2 #(.width(width)) AC(WAC,RAC,ck,ALU_out,common_bus);
type3 #(.width(width),.initial_value(initial_A)) A(Wa,Ra,ck,common_bus);
type3 #(.width(width),.initial_value(initial_B)) B(Wb,Rb,ck,common_bus);
ALU_type #(width) ALU(P1,P0,common_bus,RT_out,ALU_out);
endmodule
```

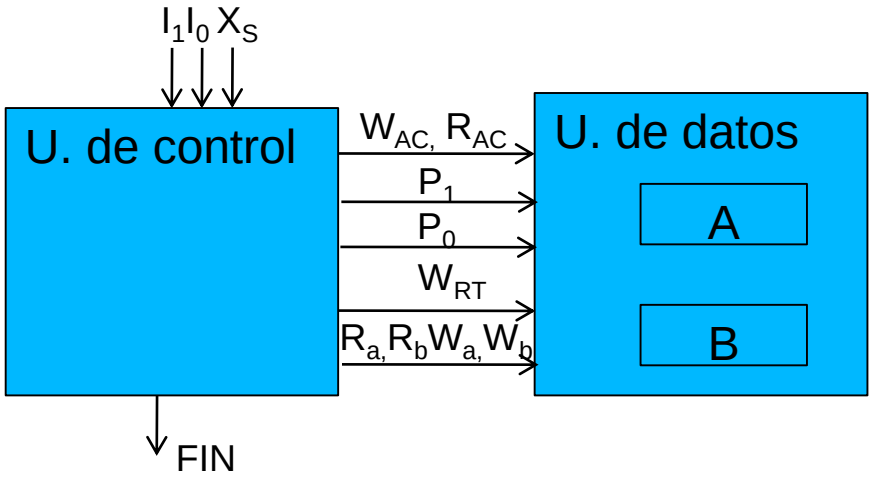
Descomposición en microoperaciones

- Paso 4 –Descomponemos las macrooperaciones en microoperaciones.
  - Durante la ejecución de una macrooperación solo pueden modificarse los registros ocultos y los registros visibles que aparezcan como destino en la descripción de la macrooperación.

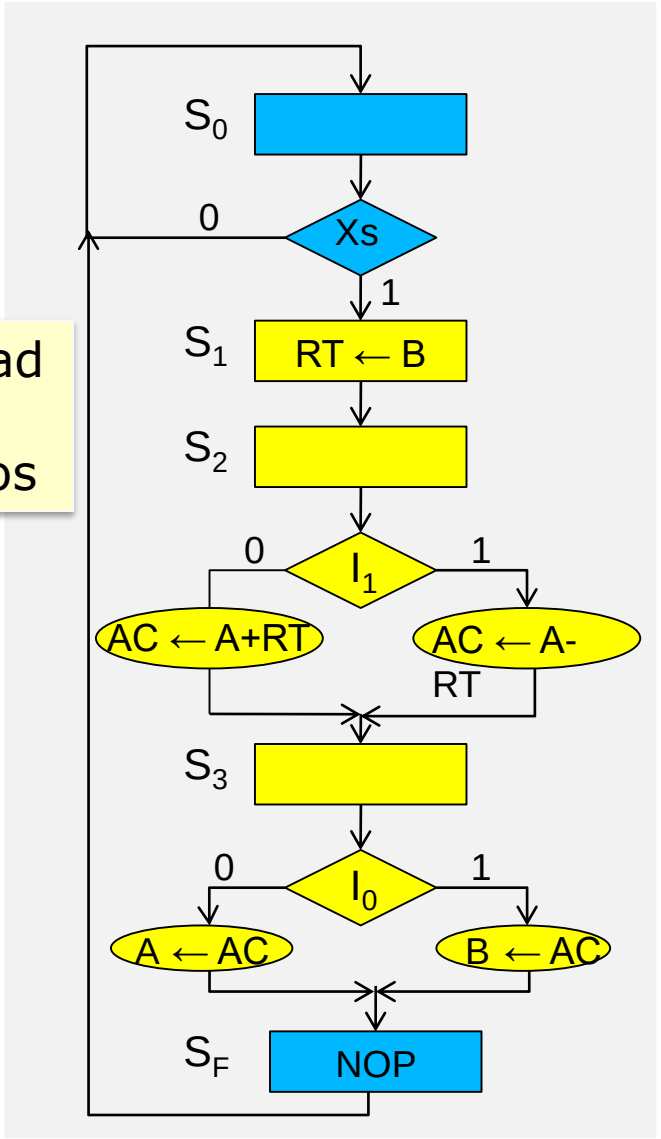
	A ← A + B	**B ← A + B**	**A ← A - B**	**B ← A - B**
μop 1	RT ← B			
μop 2	AC ← A + RT		AC ← A - RT	
μop 3	A ← AC	B ← AC	A ← AC	B ← AC

Carta ASM de la calculadora

	A ← A + B I_1I_0=00	**B ← A + B** I_1I_0=01	**A ← A – B** I_1I_0=10	**B ← A - B** I_1I_0=11
1	RT ← B			
2	AC ← A + RT		AC ← A - RT	
3	A ← AC	B ← AC	A ← AC	B ← AC

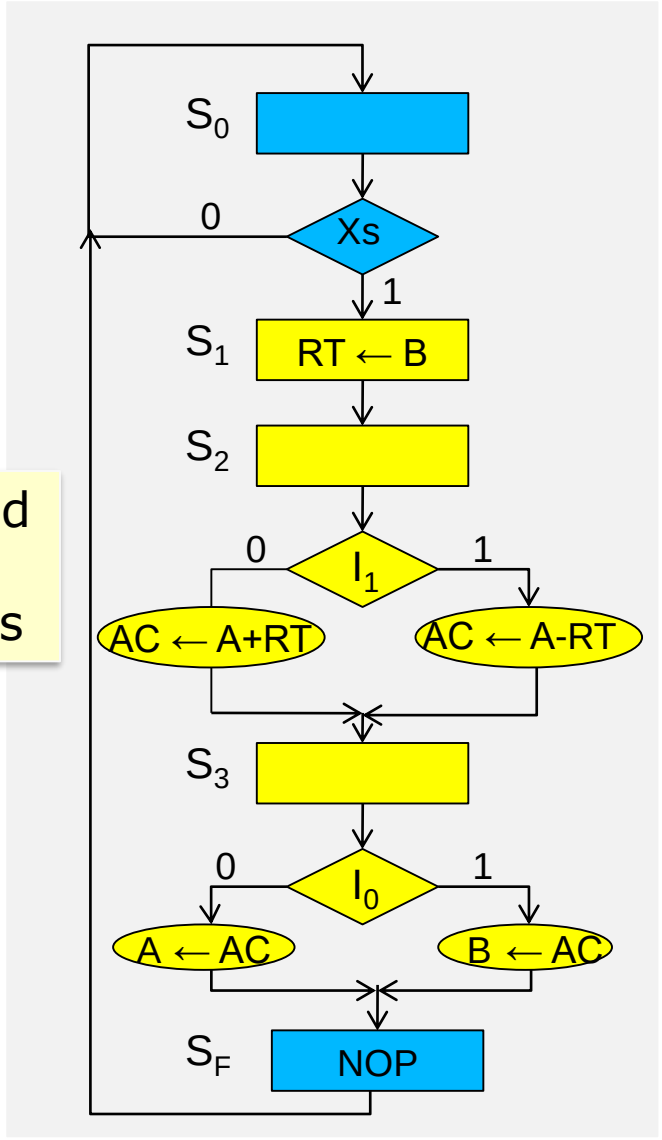


unidad de datos

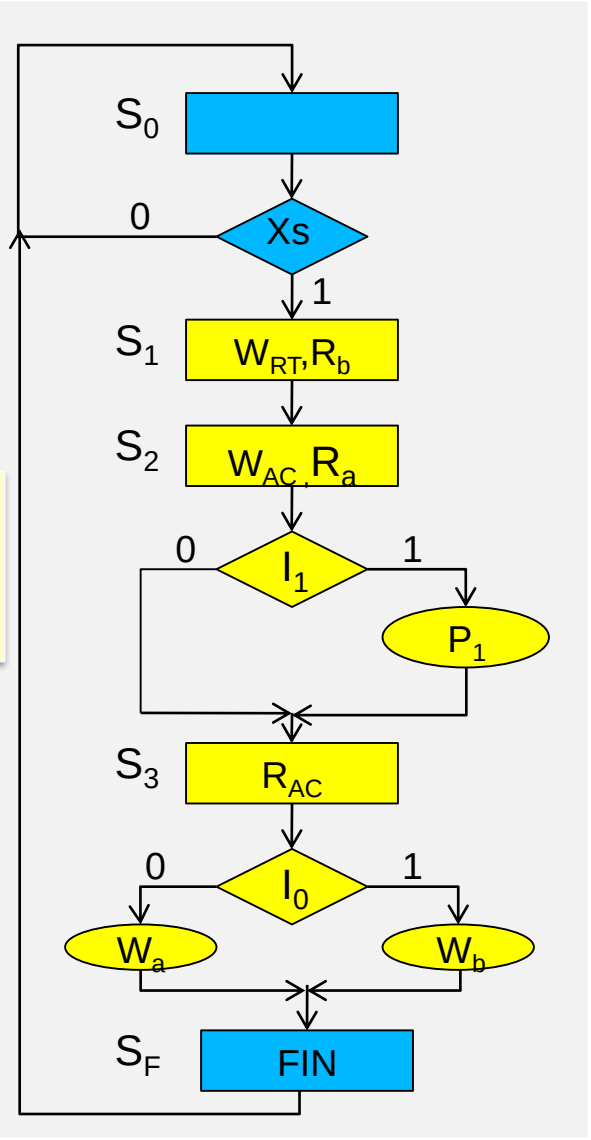


Carta ASM de la calculadora

unidad de datos



unidad de control



Descripción Verilog de la U. de control de la calculadora

- La descripción canónica de máquinas de estado en HDL Verilog es un proceso sistemático
- Se utilizará una estructura general del código en la que hay 2 procesos
  - Uno de asignación de siguientes estados
  - Otro de calculo de siguiente estado y salidas

Descripción Verilog de la u. de control de la calculadora, estructura general

```
module mi_carta_asm(
   input LISTA_DE_ENTADAS,
   output reg LISTA_DE_SALIDAS);

// DEFINICION Y ASIGNACIÓN DE ESTADOS
   parameter LISTA_DE_ESTADOS

// VARIABLES PARA ALMACENAR EL ESTADO PRESENTE Y SIGUIENTE
   reg [N:0] current_state, next_state;

// PROCESO DE CAMBIO DE ESTADO
always @(posedge clk or posedge reset)
    .......
// PROCESO SIGUIENTE ESTADO Y SALIDA
always @(current_state, LISTA_DE_ENTRADAS)
    .......
endmodule
```

Descripción Verilog de la u. de control de la calculadora, procedimiento

En la estructura general hay que completar 4 partes de código:

1. Definición y asignación de estados, según el número de estados utilizaremos mas o menos bits.

2. Definición de registros para almacenar el estado actual y el siguiente. Deben ser del mismo tamaño en bits que el utilizado en el punto anterior

3. Proceso de cambio de estado: Siempre es el mismo código

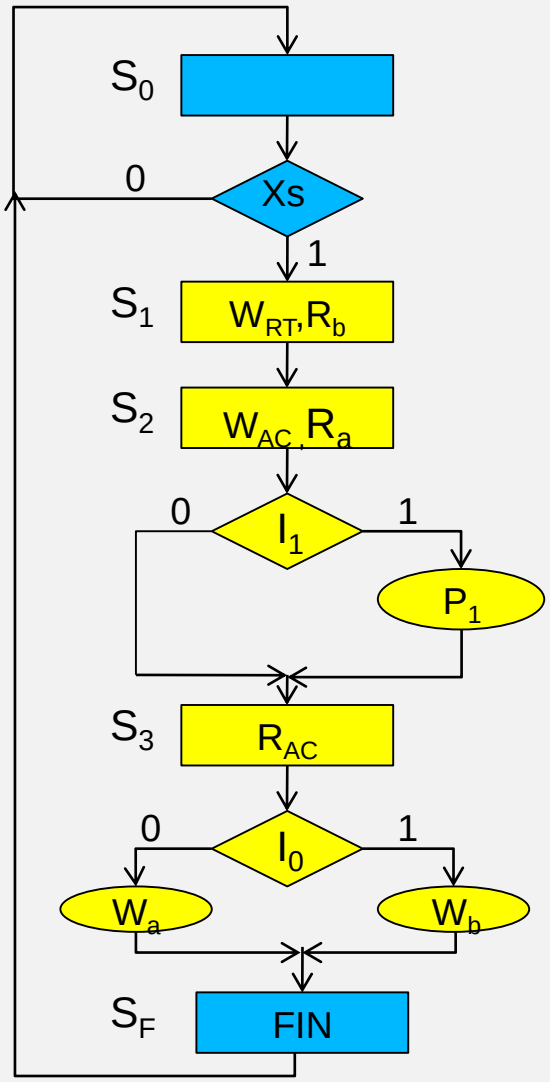
4. Proceso de cálculo de siguiente estado y salida: Hay que rellenar el código correspondiente a la carta ASM

Descripción Verilog de la U. de control de la calculadora simple

```
module carta_asm1(
    input clk, xs, i0, i1, reset,
    output reg rac, rb, ra, wrt, wac, wa, wb, p1, fin
    );
parameter S0 = 3'b000,
          S1 = 3'b001,
          S2 = 3'b010,
          S3 = 3'b011,
          SF = 3'b100;

reg [3:0] current_state,next_state;

always @(posedge clk or posedge reset)
 begin
   if(reset)
     current_state <= S0;
   else
    current_state <= next_state;
 end

                        SIGUE ->
```

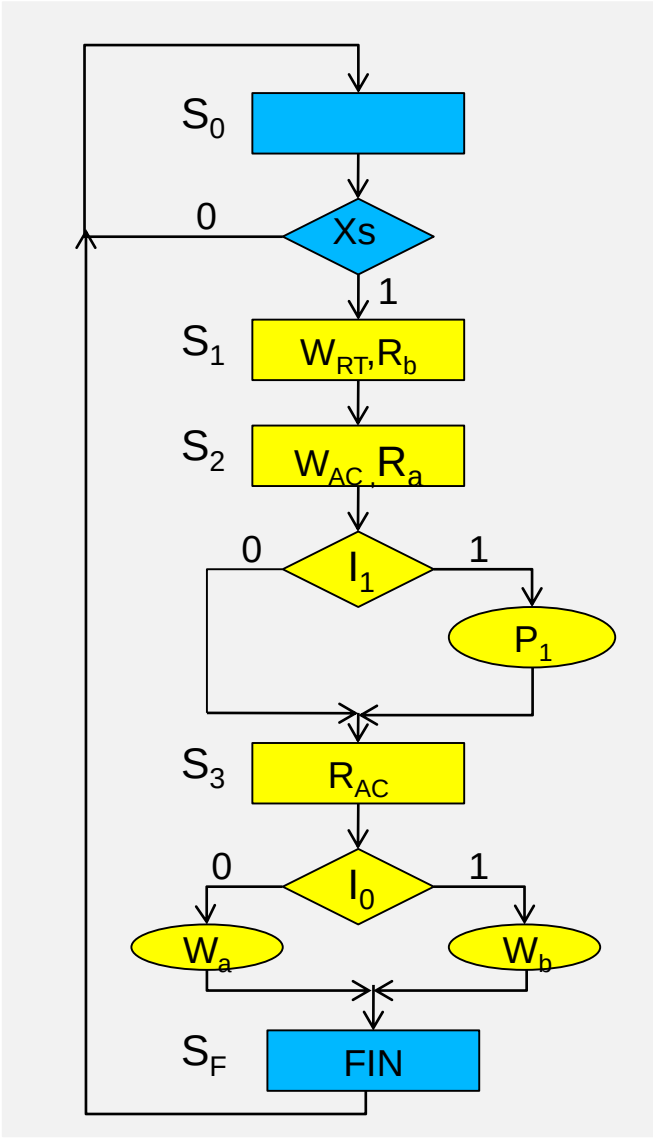
Asignación De estados

Proceso Siguiente estado



Descripción Verilog de la u. de control de la calculadora simple

- El proceso de calculo del siguiente estado y salida se realiza con una única sentencia "CASE"
  - La sentencia case debe contemplar todos los estados de la carta ASM
  - Antes de la sentencia "CASE" se recomienda establecer por defecto a cero todas las salidas.



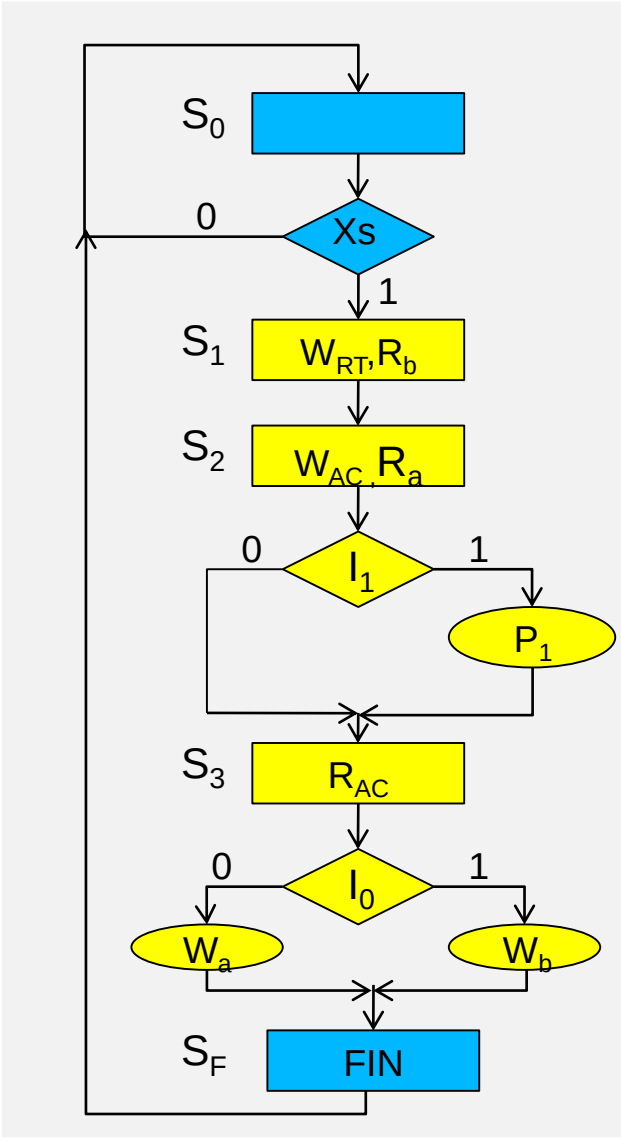
Descripción Verilog de la u. de control de la calculadora

```
always @(current_state,i0,i1)
 begin
  rac = 0;
  ra  = 0;
  rb  = 0;
  p1  = 0;
  wrt = 0;
  wa  = 0;
  wb  = 0;
  fin = 0;
  case(current_state)
   S0:
     begin
      if(xs) next_state = S1;
      else   next_state = S0;
     end
   S1:
     begin
       wrt = 1;
       rb  = 1;
       next_state = S2;
     end
```

Valor por defecto de las salidas
Establecido a cero

Estado S0

Estado S1



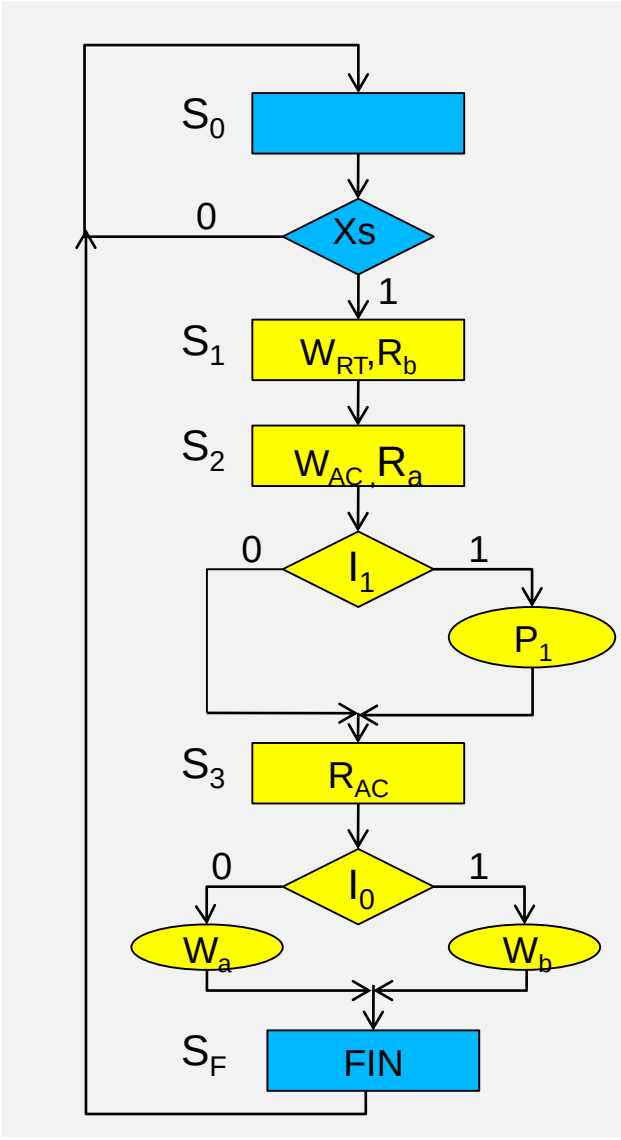
Descripción Verilog de la u. de control de la

```
S2:
   begin
      wac = 1;
      ra  = 1;
      if(i1)
           p1 = 1;
      next_state = S3;
   end
S3:
  begin
     rac = 1;
     if(i0)
       wa = 1;
     else
       wb = 1;
     next_state = SF;
  end
SF:
  begin
     fin = 1;
     next_state = S0;
  end
 endcase
end
endmodule
```

Estado S2

Estado S3

Estado SF

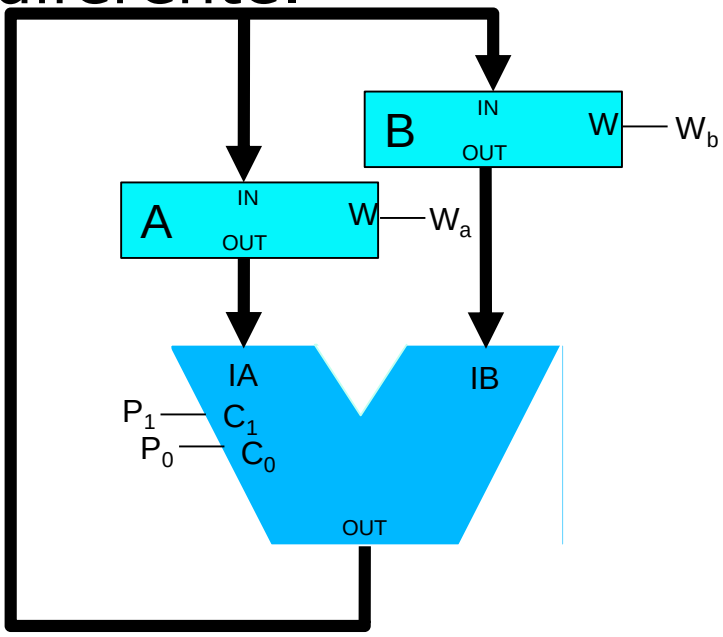


Contenidos del Tema 2

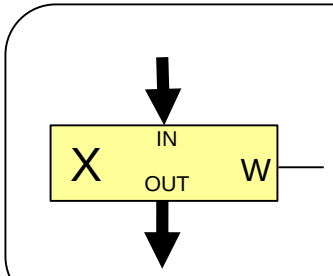
1. El nivel RT (*Register Transfer*)
2. Diseño de Sistemas Digitales
3. Diseño de la Unidad de Datos
4. Diseño de la Unidad de Control
5. Ejemplo 1: Diseño de una calculadora simple de 2 registros
6. ***Ejemplo 2: Calculadora simple (solución 3 buses)***
7. Ejemplo 3: Calculadora simple con 8 registros
8. De la Calculadora al computador

Diseño de la unidad de datos de una calculadora: solución con 3 buses

- Para las mismas especificaciones del ejemplo anterior proponemos una unidad de datos diferente.



- Arquitectura específica.
- Con esta arquitectura se necesitan menos registros.



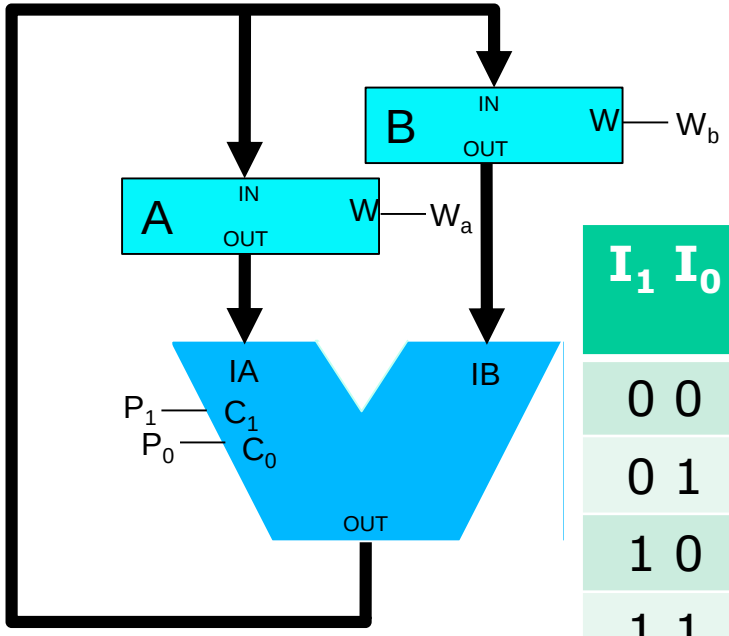
W	X ←	OUT =
0	X	[X]
1	IN	[X]

Descripción Verilog de la u. datos de la calculadora : solución con 3 buses

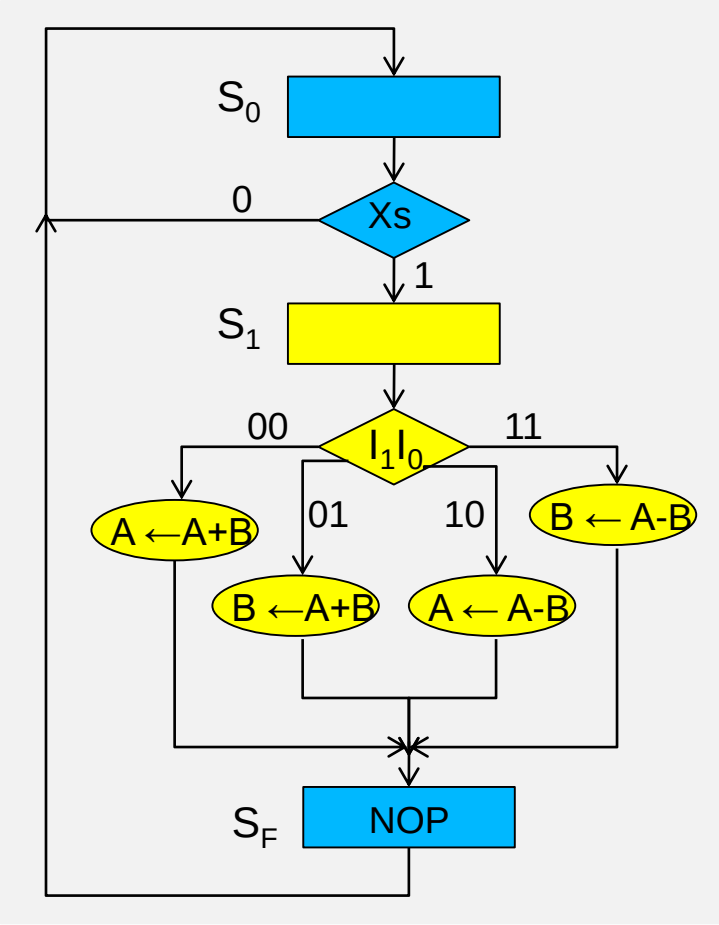
```
//declaración del tipo módulo correspondiente a RA y RB
module type1 #(parameter width=8, initial_value=0) (input wire W, ck,
                                    input wire [width-1:0] IN,
                                    output reg [width-1:0] OUT=initial_value);
        always@(posedge ck)
                if(W)
                        OUT<=IN;
endmodule
//declaración del tipo módulo correspondiente a la ALU
module ALU_type #(parameter width=8) ( input wire C1, C0,
                                    input wire [width-1:0] IA,IB,
                                    output reg [width-1:0] OUT);
        always@(*)
                case({C1,C0})
                        2'b00: OUT=IA+IB;
                        2'b01: OUT=IA;
                        2'b10: OUT=IA-IB;
                        2'b11: OUT=IB;
                endcase
endmodule
//declaración de la unidad de procesado de datos
module unidad_datos2 #(parameter width=8, initial_A=0, initial_B=1)
        (input wire ck,Wa,Wb,P0,P1);
wire [width-1:0] ALU_out, OUT_A, OUT_B;
type1 #(width,initial_A) A(Wa,ck,ALU_out,OUT_A);
type1 #(width,initial_B) B(Wb,ck,ALU_out,OUT_B);
ALU_type #(width) ALU(P1,P0,OUT_A,OUT_B,ALU_out);
endmodule
```

Carta ASM: solución con 3 buses

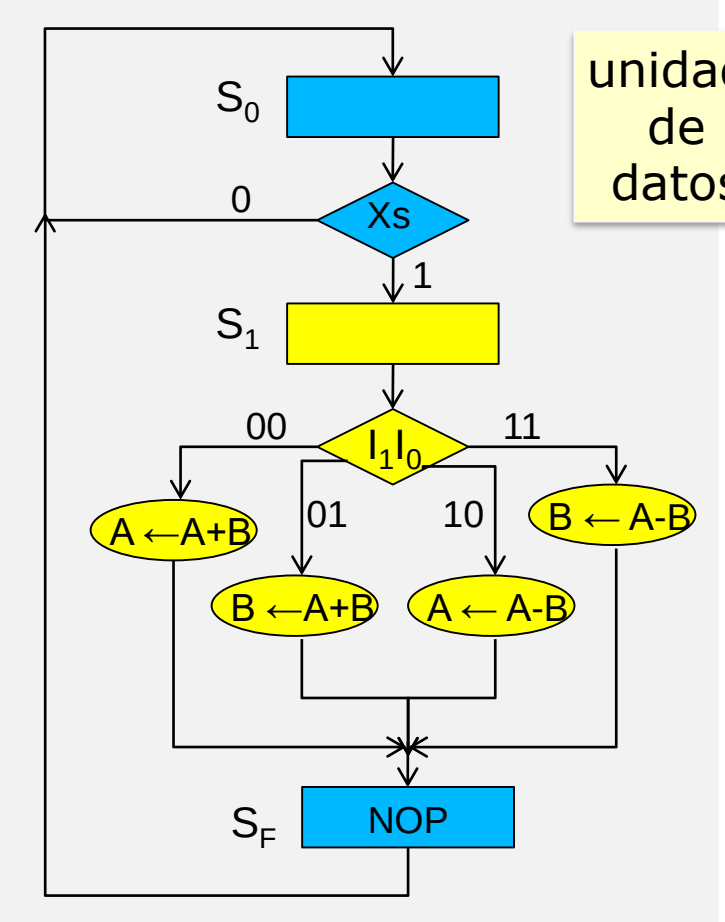
- Las macrooperaciones se realizan en un único ciclo de reloj.

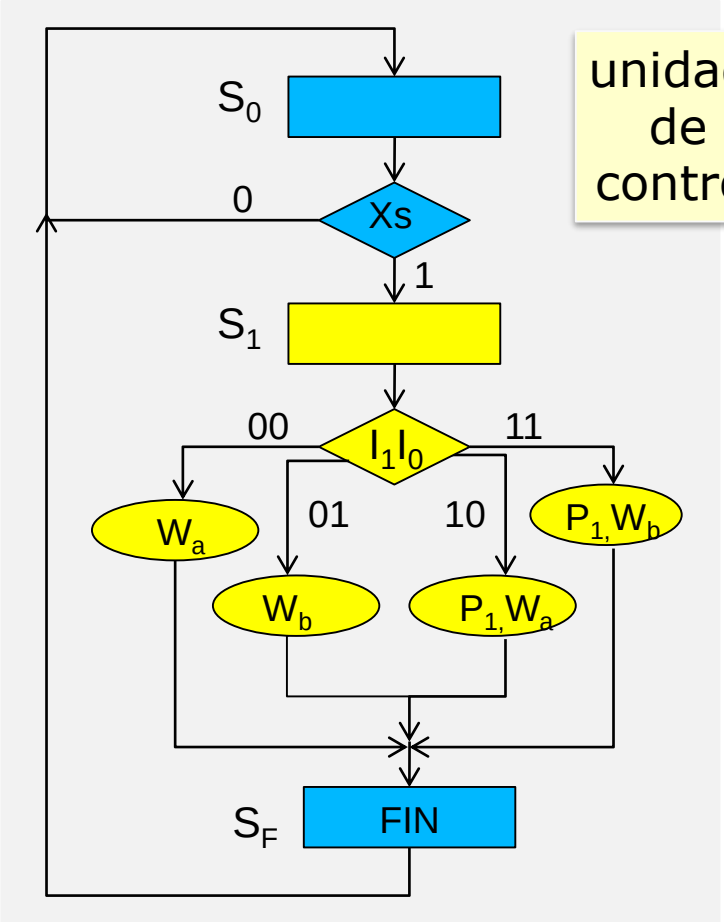


I_1 I_0	operación
0 0	A ← A + B
0 1	B ← A + B
1 0	A ← A - B
1 1	B ← A - B



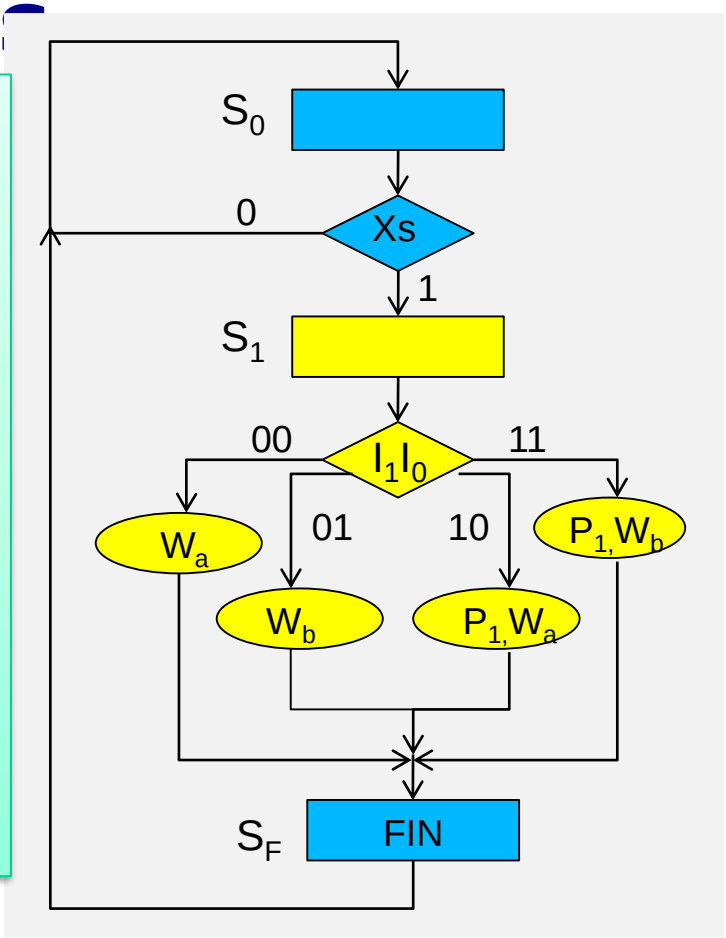
Carta ASM: solución con 3 buses





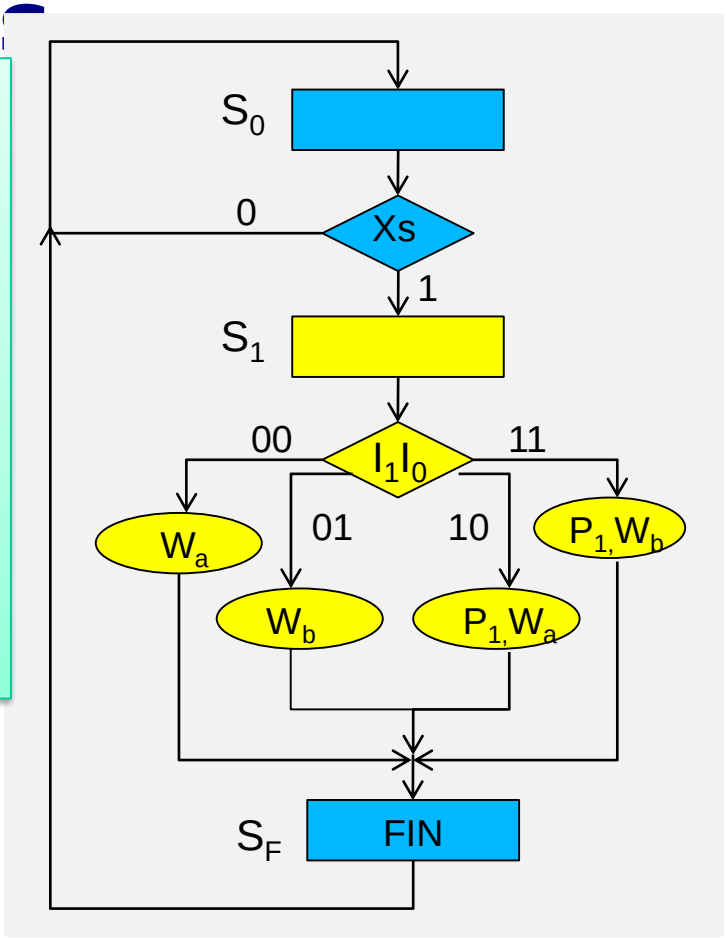
Descripción Verilog de la u. de control de la calculadora: solución con 3 buses

```
module carta_asm2(
    input clk, xs, i0, i1, reset,
    output reg wa, wb, p1, fin
    );
parameter S0 = 2'b00,
          S1 = 2'b01,
          SF = 2'b10;

reg [2:0] current_state,next_state;

always @(posedge clk or posedge reset)
 begin
   if(reset)
     current_state <= S0;
   else
    current_state <= next_state;
 end

                        SIGUE ->
```



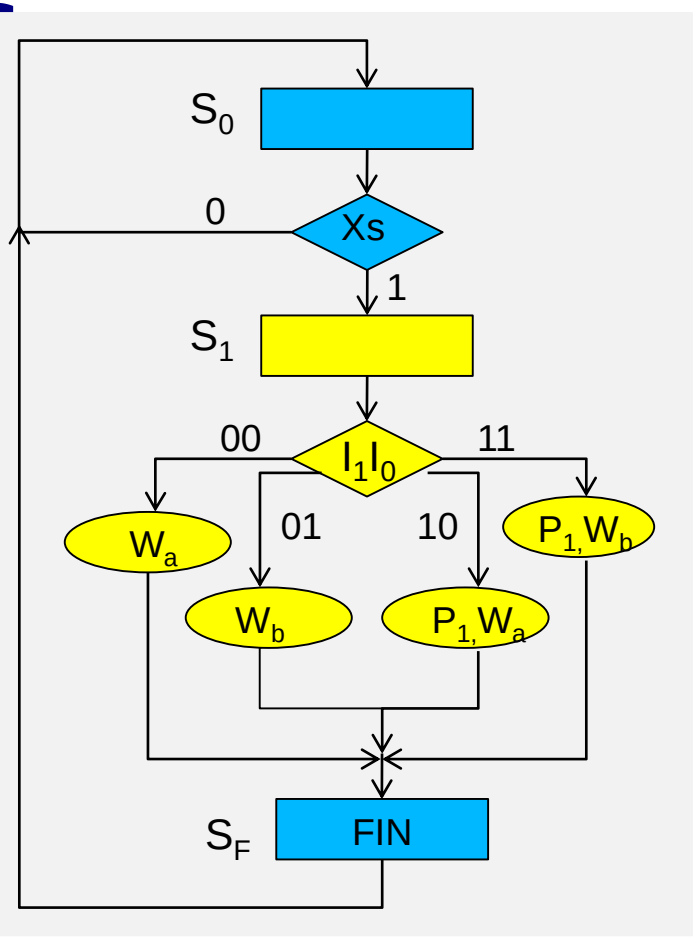
Descripción Verilog de la u. de control de la calculadora: solución con 3 buses

```
always @(current_state,i0,i1)
 begin
  p1  = 0;
  wa  = 0;
  wb  = 0;
  fin = 0;
  case(current_state)
   S0:
     begin
      if(xs) next_state = S1;
      else   next_state = S0;
     end

                        SIGUE ->
```



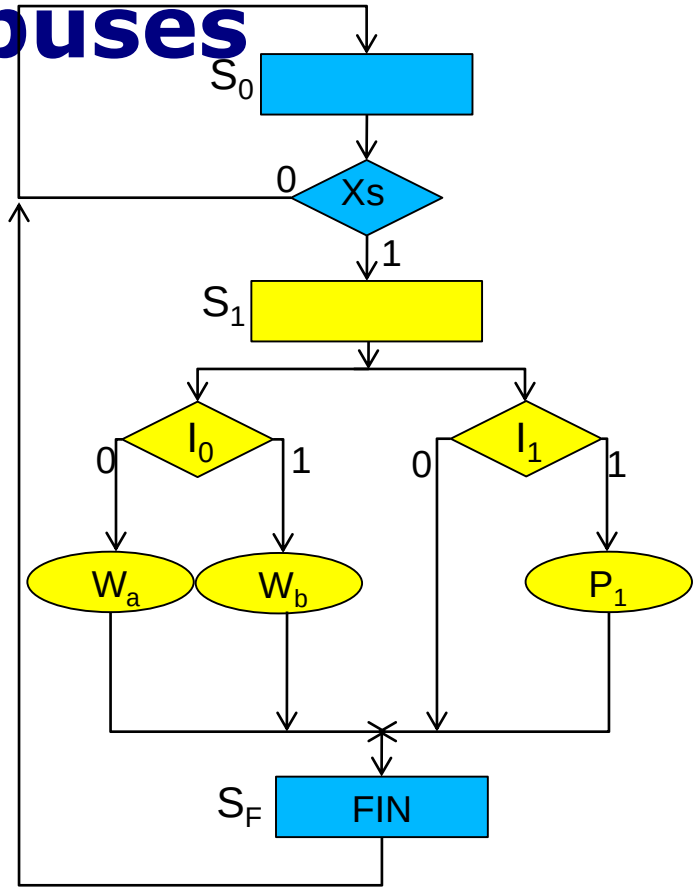
Descripción Verilog de la u. de control de la calculadora: solución con 3 buses

```
S1:
  begin
    if(i1==0 && i0==0)
        wa = 1;
      else if(i1==0 && i0==1)
        wb = 1;
      else if(i1==1 && i0==0)
        begin
         p1 = 1;
           wa = 1;
         end
      else
        begin
          p1 = 1;
          wb = 1;
         end
    next_state = SF;
  end
SF:
   begin
    fin = 1;
    next_state = S0;
   end
endcase
end
```

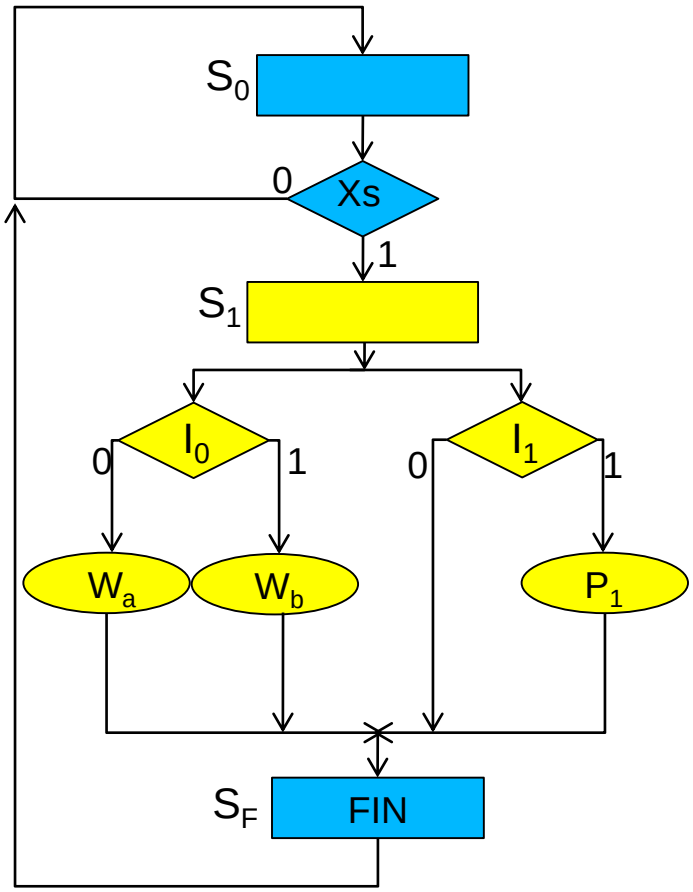


Descripción Verilog (compacta) de la u. de control de la calculadora: solución con 3 buses

```
module unidad_control_2(
    input clk, xs, i0, i1, reset,
    output reg p0,p1,wa,wb,fin
    );
parameter S0 = 2'b00,
          S1 = 2'b01,
          SF = 2'b10;

reg [2:0] current_state,next_state;

always @(posedge clk or posedge reset)
 begin
   if(reset)
     current_state <= S0;
   else
    current_state <= next_state;
 end

                    SIGUE ->
```



```
always @(current_state,i0,i1)
 begin
  p0 = 0;
  p1 = 0;
  fin = 0;
  wa = 0;
  wb = 0;
  case(current_state)
   S0:
     begin
      if(xs) next_state = S1;
      else   next_state = S0;
     end
    S1:
     begin
       if(i0)
         wb = 1;
       else
         wa = 1;
       if(i1)
         p1 = 1;
       next_state = SF;
     end
    SF:
      begin
       fin = 1;
       next_state = S0;
      end
  endcase
 end
endmodule
```



Diseño de una calculadora con 8 registros

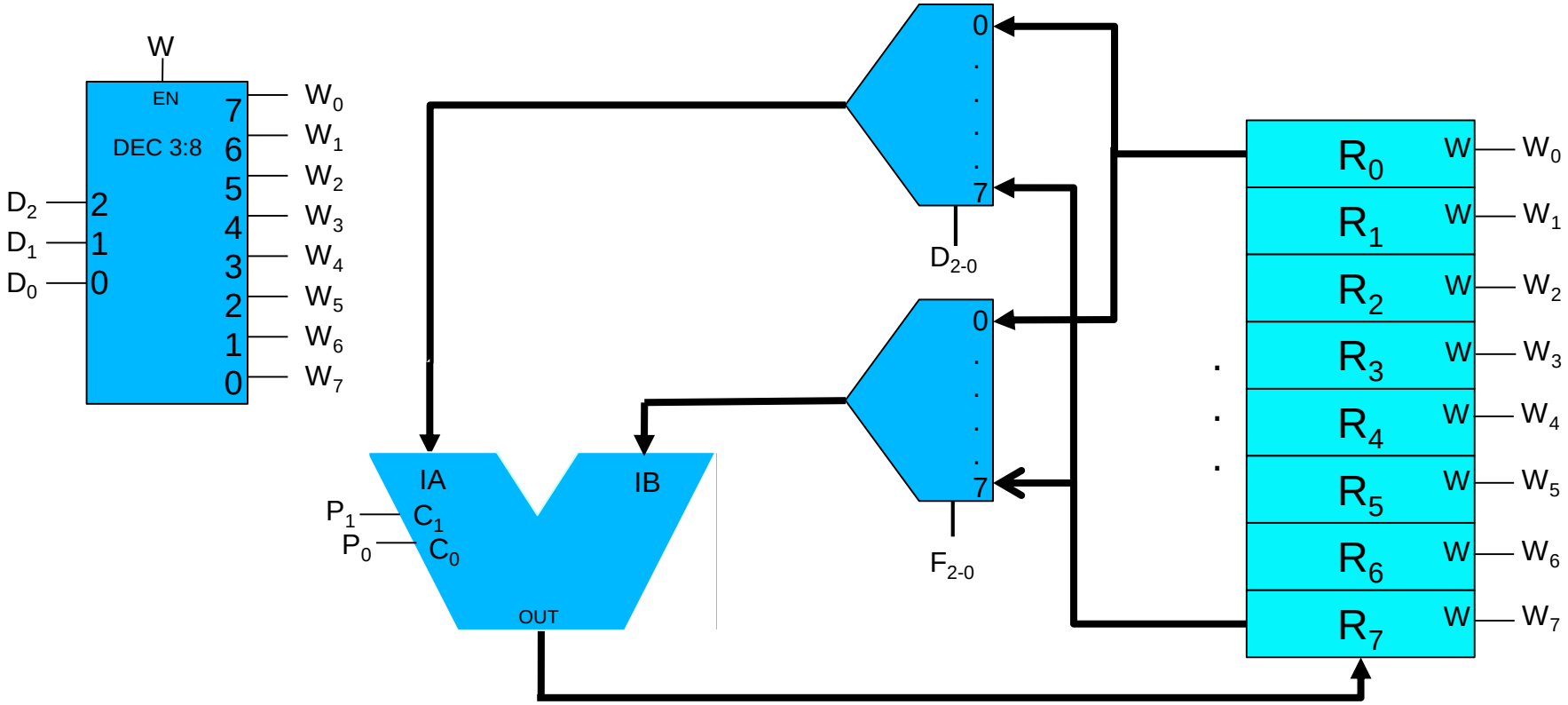
- Especificaciones del sistema a diseñar:
  - Se dispone de 8 registros (R_0, R_1, …, R_7) y se desea poder realizar cualquiera de las siguientes operaciones:

$I_1 I_0$	operación
0 0	$R_D \leftarrow R_D + R_F$
1 0	$R_D \leftarrow R_D - R_F$
0 1	$R_D \leftarrow R_F$

  - D,F $\in$ {0,1,2,…,6,7}
  - D y F vienen determinados por ($D_2 D_1 D_0$) y ($F_2 F_1 F_0$)

Diseño de una calculadora con 8 registros

- Arquitectura de la u. de datos:



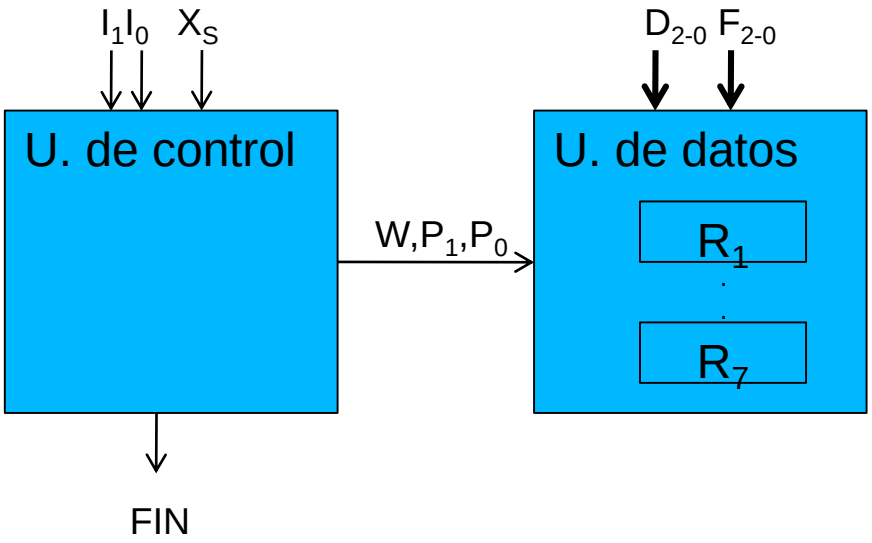
Diseño de una calculadora con 8 registros

- Descripción Verilog de la unidad de procesado

```
//declaración del tipo módulo correspondiente a la ALU
module ALU_type #(parameter width=8) ( input wire C1, C0,
                                       input wire [width-1:0] IA,IB,
                                       output reg [width-1:0] OUT);
        always@(*)
                case({C1,C0})
                        2'b00: OUT=IA+IB;
                        2'b01: OUT=IA;
                        2'b10: OUT=IA-IB;
                        2'b11: OUT=IB;
                endcase
endmodule
//declaración de la unidad de procesado de datos
module unidad_datos3 #(parameter width=8, initial_value_R0=1)
        (input wire ck,W,P0,P1, input wire [2:0] F,D);

        wire [width-1:0] ALU_out;
        reg [width-1:0] R [7:0];
        ALU_type #(width) ALU(P1,P0,R[D],R[F],ALU_out);

        always@(posedge ck)
                if(W)
                        R[D]<=ALU_out;

        initial
                R[0]<=initial_value_R0;//para testear
endmodule
```

Diseño de una calculadora con 8 registros

- Organización del sistema digital:

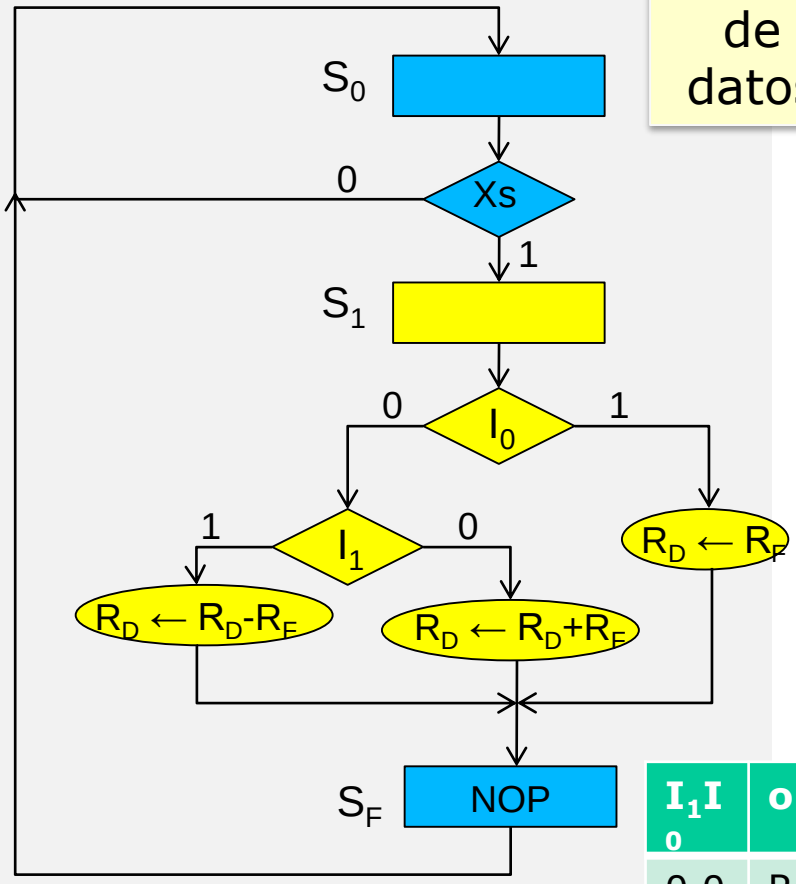


- El usuario especifica la operación proporcionando el valor de I_1, I_0, D_{2-0}, F_{2-0}, y genera la orden de comienzo con X_S

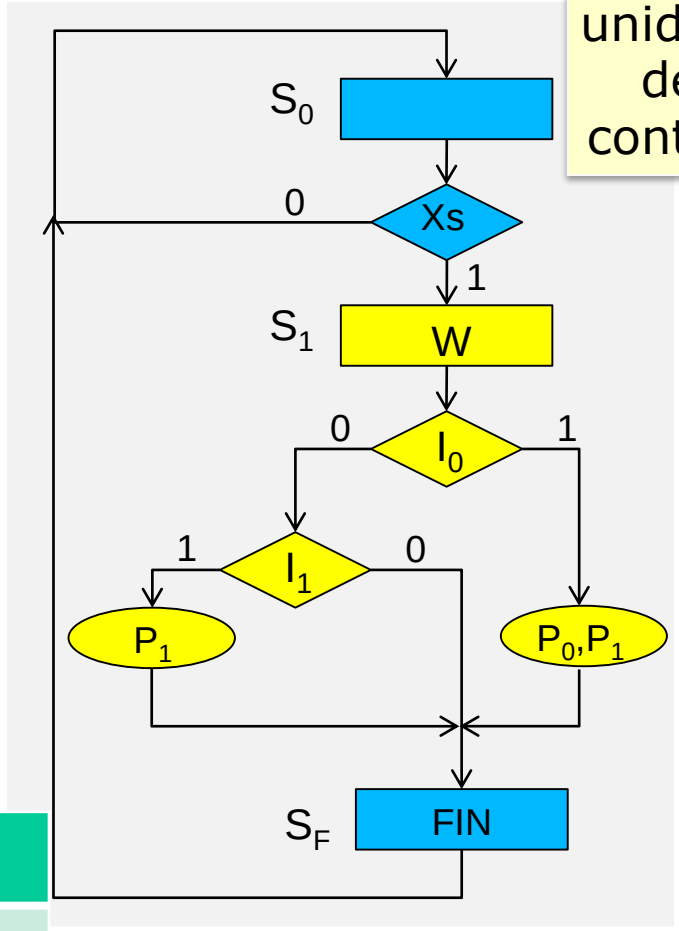
Diseño de una calculadora con 8 registros

- Carta ASM

unidad de datos



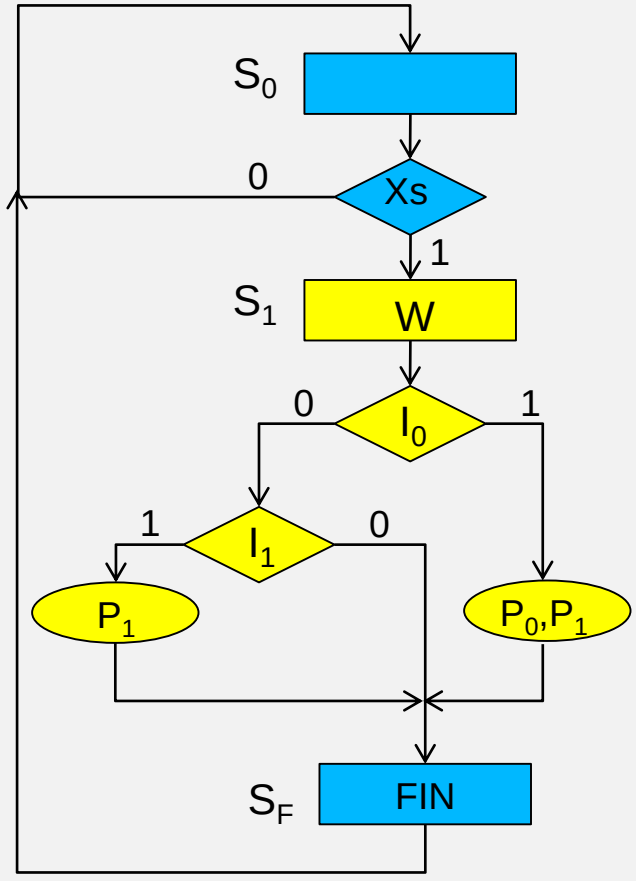
unidad de control



I_1I_0	operación
0 0	$R_D \leftarrow R_D + R_F$
1 0	$R_D \leftarrow R_D - R_F$
0 1	$R_D \leftarrow R_F$

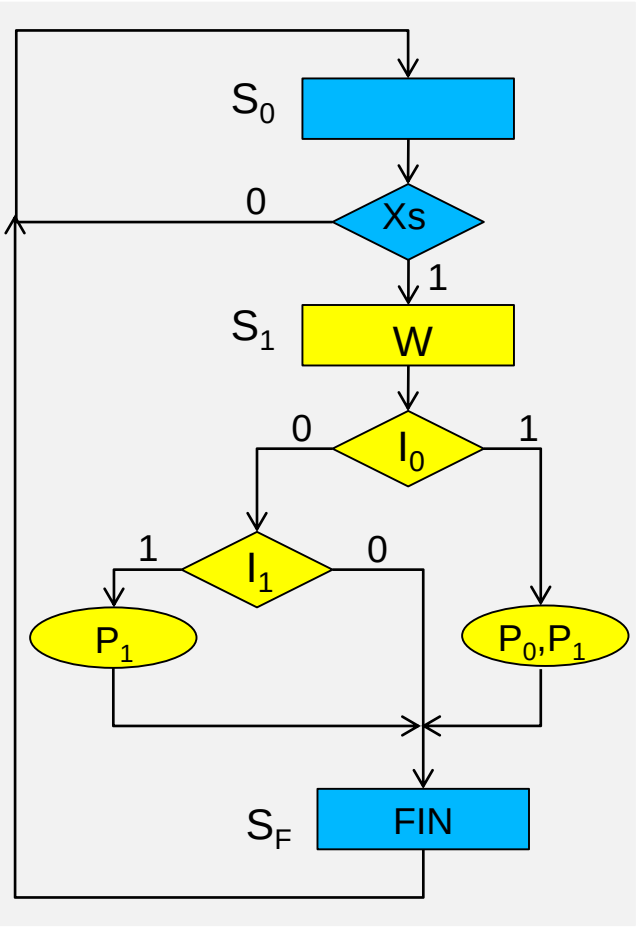
Diseño de una calculadora con 8 registros

- Carta ASM y descripción Verilog del controlador




```
module unidad_control_3(
     input clk, xs, i0, i1, reset,
     output reg p0,p1,w,fin
     );
parameter S0 = 2'b00,
          S1 = 2'b01,
          SF = 2'b10;

reg [2:0] current_state,next_state;

always @(posedge clk or posedge reset)
 begin
   if(reset)
     current_state <= S0;
   else
    current_state <= next_state;
 end
```

SIGUE ->




```
always @(current_state,i0,i1)
 begin
  p0 = 0;
  p1 = 0;
  fin = 0;
  w = 0;
  case(current_state)
   S0:
     begin
      if(xs) next_state = S1;
      else   next_state = S0;
     end
    S1:
     begin
         w = 1;
       if(i0)
         begin
          p0 = 1;
          p1 = 0;
         end
       else if (i1)
         p1 = 1;
       next_state = SF;
     end
    SF:
      begin
       fin = 1;
       next_state = S0;
      end
  endcase
 end
endmodule
```

Ejemplo de uso de la calculadora

- Realización de la operación $R_0 \leftarrow 3R_1 - R_2$
  - Se trata de una operación más compleja no incluida en la tabla de operación del sistema.
  - Se puede realizar mediante una secuencia de instrucciones (nivel ISP)
    - $R_0 \leftarrow R_1$
    - $R_0 \leftarrow R_0 - R_2$
    - $R_0 \leftarrow R_0 + R_1$
    - $R_0 \leftarrow R_0 + R_1$

Calculadora frente a computador

- Similitudes
  - Podemos resolver problemas complejos a partir de las instrucciones del sistema mediante programación (software).
  - El usuario no necesita ser especialista en la electrónica del sistema (hardware).
- Deficiencias
  - No hay automatización en la ejecución del programa: cada vez que se ejecuta una instrucción el usuario debe activar Xs, esperar la señal de FIN y suministrar la siguiente.
  - No hay programa almacenado: para ejecutar cada instrucción el usuario debe proporcionar los valores $D_{2:0}$ y $F_{2:0}$ para cada una de las tres instrucciones.